SCENIX

User's Manual _{TM} Virtual Peripheral Methodology & Modules

Revision History

REVISION	RELEASE DATE	SUMMARY OF CHANGES
0.998	February 24, 2000	Initial release (Word version)
1.0	March 24, 2000	Reformatted for distributing the initial draft.

©2000 Scenix Semiconductor, Inc. All rights reserved. No warranty is provided and no liability is assumed by Scenix Semiconductor with respect to the accuracy of this documentation or the merchantability or fitness of the product for a particular application. No license of any kind is conveyed by Scenix Semiconductor with respect to its intellectual property or that of others. All information in this document is subject to change without notice.

Scenix Semiconductor products are not authorized for use in life support systems or under conditions where failure of the product would endanger the life or safety of the user, except when prior written approval is obtained from Scenix Semiconductor.

ScenixTM and the Scenix logo are trademarks of Scenix Semiconductor, Inc.

Virtual PeripheralTM is a trademark of Scenix Semiconductor, Inc.

I²CTM is a trademark of Philips Corporation

MicrowireTM is a trademark of National Semiconductor Corporation

All other trademarks mentioned in this document are property of their respective companies.

Scenix Semiconductor, Inc., 1330 Charleston Road, Mountain View, CA 94043, USA Telephone: +1 650 210 1500, Fax: +1 650 210 8715, Web site: www.scenix.com,

E-mail: sales@scenix.com

Contents

Chapter 1	Virtual Peripheral Guidelines	
1.1	Introduction	4
1.2	Virtual Peripheral Implementation	
1.3	Structure of an Application with Virtual Peripheral Modules	
1.4	RAM	
1.5	Labels	
	1.5.1 Labeling All Sections: RAM, Constants, ISR, Subroutines	
1.6	Standardized Directives	
1.7	Standard !Option Setup	9
1.8	Standard Mode Register Setup	10
1.9	Port Access	10
1.10	Port Direction	11
1.11	SX28AC/SX52BD Compatibility	12
1.12	Lookup Tables (Jump Tables)	14
1.13	Memory Location Dependencies	15
1.14	Page Boundaries	16
1.15	Subroutines in the Second Page of Program Memory	16
1.16	Defining ORG Statements	18
1.17	Main Program	
1.18	Making Frequency Scalable	19
1.19	Making Virtual Peripheral Modules Look and Act as Modules	21
1.20	Define the ISR in Locations \$0 to \$1ff	21
1.21	Worst-Case ISR Cycle Count	21
1.22	Selecting the Interrupt Rate and Oscillator Frequency	22
1.23	Saving CPU Bandwidth	22
1.24	Use the Multi-Threaded ISR Template	23
Chapter 2	Source Code Template	
2.1	Introduction	27
2.1	Source Code Template	27
Chapter 2	Adding a Virtual Peripheral to the Source Code Template	
3.1	Introduction	48
3.2	Source Code Template With A Virtual Peripheral Example	48

SCENIX

Chapter 1

Virtual Peripheral Guidelines

1.1 Introduction

This document describes the formats, conventions, and coding guidelines for developing Virtual Peripheral modules and integrating the modules for an application that uses the SX communications controller. By applying a familiar model consistently across all Virtual Peripheral modules, we can keep the design methodology simple to reduce the development effort.

Virtual Peripheral concept enables the "software system on a chip" approach. Virtual Peripheral, a software module that replaces a traditional hardware peripheral, takes advantage of the Scenix architecture's high performance and deterministic nature to produce same results as the hardware peripheral with much greater flexibility.

The speed and flexibility of the Scenix architecture complemented with the availability of the Virtual Peripheral library, simultaneously address a wide range of engineering and product development concerns. They decrease the product development cycle dramatically, shortening time to production to as little as a few days.

Scenix's time-saving Virtual Peripheral library gives the system designers a choice of ready-made solutions, or a head start on developing their own peripherals. So, with Virtual Peripheral modules handling established functions, design engineers can concentrate on adding value to other areas of the application.

The concept of Virtual Peripheral combined with in-system re-programmability provides a powerful development platform ideal for the communications industry because of the numerous and rapidly evolving standards and protocols.

Overall, the concept of Virtual Peripheral provides benefits such as using a more simple device, reduced component count, fast time to market, increased flexibility in design, customization to your application and ultimately overall system cost reduction.

Some examples of Virtual Peripheral modules are:

- Communication interfaces such as I²CTM, MicrowireTM (μ-Wire), SPI, IrDA Stack, UART, and Modem functions
- Internet Connectivity protocols such as UDP, TCP/IP stack, HTTP, SMTP, POP3
- Frequency generation and measurement
- PWM/PDM generation
- Delta/Sigma ADC
- DTMF generation/detection
- FFT/DFT based algorithms

1.2 Virtual Peripheral Implementation

The concept of Virtual Peripheral has been around for many years, but the hardware to make the concept practical did not exist until recently when Scenix released the SX communications controller. The SX communication controller is a cost-effective communications controller, running at up to 100 MIPS. Although speed is an important feature of the SX communication controller, its deterministic architecture is the essential enabling technology for Virtual Peripheral implementation. Every instruction in the SX is completely deterministic, in the sense that all instructions execute in a predetermined number of clock cycles (1 cycle for all instructions except branches, which require 3 cycles) and the interrupt latency is fixed (3 cycles for an internal interrupt, 5 cycles for an external interrupt).

All Virtual Peripheral modules run in the "background" of the main application software, as part of an interrupt service routine. The deterministic nature of the SX communication controller gives the interrupt service routine an exact frequency of execution, and all Virtual Peripheral modules, whether a serial bus interface, a timer, or a DTMF generator, can be based on this exact, jitter-free frequency.

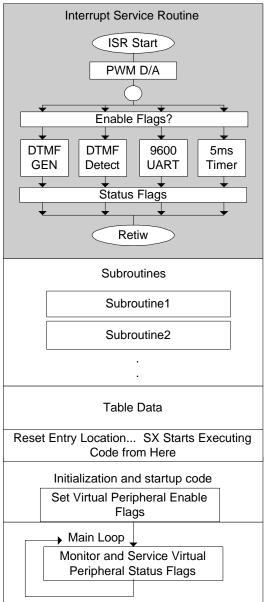
Virtual Peripheral modules must run with minimal intervention from the application software, just as though they were hardware peripherals. The application software simply sets or clears flags, loads a few registers, and then lets the Virtual Peripheral modules do the work. For example, with an A/D Virtual Peripheral, there is no interaction needed from the application software while the conversion is taking place. When the A/D Virtual Peripheral has finished a conversion, it can set a flag to indicate that the application can take the result.

The interrupt service routine is set up to be called at an exact timing interval. For most Virtual Peripheral modules, the timing can be arbitrary, as long as the sample rate is high enough to accomplish the desired task. Some Virtual Peripheral modules such as UART needs to run at specific rates, e.g. 1200 baud, 115.2 kbaud, etc. To simplify the interrupt service routine code that performs the UART function, the interrupt service routine is called at a rate that is an exact multiple of the standard UART speeds.

Virtual Peripheral modules for the SX communication controller are usually designed to run from interrupts triggered by the Real-Time Clock/Counter (RTCC). The RETIW instruction performs a return from interrupt that also adjusts the RTCC to control the exact timing of the interrupts. In the Scenix design methodology, the programmer creates a real-time kernel that receives the interrupts and allocates execution time to the various Virtual Peripheral modules that may run.

The main body of the application starts execution at the reset vector, initializes the system, then falls into the main loop. The main loop communicates with the Virtual Peripheral modules through flags. Enable flags signal when a Virtual Peripheral should run, e.g. to request transmitting a character. Status flags indicate when a Virtual Peripheral has completed an operation, e.g. to acknowledge that the character has been transmitted. The main loop does not handle interrupts directly, that function being handled entirely in background by the real-time kernel.

1.3 Structure of an Application with Virtual Peripheral Modules



The Interrupt Service Routine calls Virtual Peripheral that run transparently to the main program (i.e. in background). The ISR is called at a specific frequency and services all of the Virtual Peripheral modules.

In this illustration, a PWM D/A converter is running as a fixed-rate, high-priority task, and four other tasks (DTMF generation, DTMF detection, UART, and timer) are running at lower priority controlled by enable flags.

The Virtual Peripheral modules set status flags to indicate their status.

The subroutines follow the end of the interrupt service routine.

Table data like strings can be stored after the subroutines.

The entry point for the application (where the program begins running on reset) is before the main loop at the end of the source code.

The main loop of the application is located at the end of the source code. It sets enable flags to start Virtual Peripherals running in the Interrupt Service Routine. Virtu Peripherals set status flags to indicate completion, such as the 5 ms timer expiring.

Figure 1-1 Virtual Peripheral Implementation

1.4 **RAM**

Use the following standard label names for global variables in Virtual Peripherals and applications released by Scenix:

flags0	equ	global_org+0	<pre>; semaphore register reserved ; for flag bits.</pre>
isrTemp0	equ	global_org+1	<pre>; temporary register reserved ; for use by the interrupt- ; service routine, ISR Virtual Peripheral</pre>
localTemp0	equ	global_org+2	<pre>; temporary register reserved ; for use by subroutines</pre>
localTemp1	equ	global_org+3	<pre>; temporary register reserved ; for use by subroutines</pre>
localTemp2	equ	global_org+4	<pre>; temporary register reserved ; for use by the main program</pre>

flags0 stores bitwise operators like flags and function-enabling bits (semaphores).

isrTemp0 is for use ONLY by the interrupt service routine as a global register.

localTemp0 is the temporary register used most often, so routines that are only nested once can destroy this register. It is never guaranteed to retain data from routine to routine.

localTemp1 is used by the second nested level, or when a routine needs more than one temporary global register.

localTemp2 follows along the same lines as localTemp1, but is used even less often by more deeply nested routines or as a mainline loop counter, because the other temporary registers will probably be destroyed by the routines called by the mainline.

The documentation for each subroutine will specify which localTemp register it destroys, and which localTemp registers are destroyed by routines nested below this one. If additional temporary registers are needed, they can be called localTemp3 or flags1, etc.

Write Virtual Peripheral modules so they make use of no global RAM locations other than the definitions listed above.

1.5 Labels

All labels must be kept under two tabs in length. The Hungarian notation must be used for all labels. Example:

RS232 receive becomes rs232Receive

Prefix all RAM locations and constants for a specific Virtual Peripheral by a standard, truncated version of that Virtual Peripheral's name. Example:

rxByte ds 1 becomes rs232RxByte ds 1

Left justify all equates and defines, and group all of them into the "Equates and Definitions" area of the source code.

1.5.1 Labeling All Sections: RAM, Constants, ISR, Subroutines

- Each section of the Virtual Peripheral must be *clearly labeled* to show which Virtual Peripheral it belongs to. This includes RAM allocations, program constants, Virtual Peripheral (ISR) routines, and callable subroutines.
- The labels should be machine readable, to enable a smart software tool in the future that can automatically cut and paste Virtual Peripheral modules.
- Start the Virtual Peripheral module with:

```
;VP_begin <moduleVP>
```

And end it with:

```
;VP_end ; End cut/paste <module VP>
```

```
;VP_begin RS232Receive

RS232RxBank = $

RS232RxCount ds 1 ;number of bits received

RS232RxDivide ds 1 ;receive timing counter

RS232RxByte ds 1 ;buffer for incoming byte

;VP_end ; End cut/paste for RS232Receive
```

1.6 Standardized Directives

Create a set of standard directives for all Virtual Peripheral modules. A tradeoff is that some Virtual Peripheral modules may become less code-efficient or speed-efficient.

Standard Directives:

- OPTIONX enabled
- STACKX enabled
- CARRYX disabled if a routine cannot run without CARRYX (like hard-core math), the fact that CARRYX is necessary must be well documented.
- TURBO mode

1.7 Standard !Option Setup

- WREG enabled (bit seven of !OPTION = 0, !OPTION = 0xxxxxxxb), so W is accessible as a file register in location \$01, rather than RTCC.
- If possible, a routine that needs to access the RTCC register in location \$01 should set bit 7 of !OPTION before executing and, when finished, clear it again to access the WREG in location \$01.

Define the default !option set-up as

```
OPTIONSETUP equ RTCC_PS_OFF|PS_111 ;!OPTION = 0000111b
```

Initialization code for WREG in location \$01:

```
mov w,#OPTIONSETUP ; setup option register for RTCC interrupts
mov !option,w ; enabled and no prescaler.
jmp @main
```

Routines accessing the RTCC register in location \$01:

```
mov
                  !option, #(OPTIONSETUP | RTCC_ON) ; enable direct access of the RTCC
                                                    ; by setting !option.7
                  !rb,RTCC
           mov
                                                    ; This code accomplishes absolutely
                                                    ; nothing,
                  :elsewhere
           jmp
                                                    ; but it accesses the RTCC register
:elsewhere mov
                 RTCC,rc
                  !option, #OPTIONSETUP
           mov
                                                    ; go back to the option register's
                                                    ; default.
```

After accessing the RTCC register, this routine sets the option register back to its default state. If an exception must be made for speed purposes, it should be well documented that the routine needs the option register set up to allow direct access to the RTCC.

1.8 Standard Mode Register Setup

In mainline (interruptible) code, never assume the value in the mode register, and always update it before using it.

In the Interrupt Service Routine, routines changing the mode register must change it back before exiting. The isrTemp register can be used to store and restore the previous state of the mode register. Example:

```
; save mode register in isrTemp
mov
mov
             isrTemp,w
mov
             w,#$1f
                                ; change mode register
mov
             m,w
             !rb,#0
                                ; change port RB to all outputs
mov
mov
             w,isrTemp
                                ; restore mode register
mov
             m,w
```

1.9 Port Access

To ease integrating multiple Virtual Peripheral modules that all need access to the same ports,

- Use pin definitions rather than port definitions.
- All port accesses should be made through symbolic names. Example:

setb rb.6 becomes setb LEDPin

1.10 Port Direction

When a Virtual Peripheral must dynamically change a port direction register, it should do this through the use of a *port direction buffer*. The port direction register stores the initialized state of the port direction register, and any changes made to the port direction register are made to the buffer first, and then the buffer is written to the port direction register.

If two Virtual Peripheral modules are combined, and both need to dynamically modify the same port direction register, they instead operate on the buffer for that port's direction register. The buffer is then written to the port direction register. Use banked RAM and standardized names for port direction register buffers:

portBufBank	equ	\$
RADirBuf	ds	1
RBDirBuf	ds	1
RCDirBuf	ds	1
RDDirBuf	ds	1
REDirBuf	ds	1

These rules apply to other special mode-register addressable registers, such as the pull-up enable registers, etc.

1.11 SX28AC/SX52BD Compatibility

Use MACROs or IFDEF/IFNDEF statements to make portions of incompatible code switch in and out for the SX28AC and SX52BD communications controllers.

Example of using an IFDEF statement for SX18/28AC portability to SX52BD and vice-versa:

```
; Virtual Peripheral: 62-byte buffer
; Subroutine: Store W in buffer[pushIndex++]
              INPUTS: data to store in W
OUTPUTS: data to store in buffer[pushIndex++]
CHANGES: localTemp1,pushIndex,buffer[pushIndex]
bufferPush
                 localTemp1,w
        mov
        _bank buffer
        mov fsr,pushIndex
                 indf,localTemp1
         _bank buffer
                  pushIndex
         inc
IFDEF
        SX28AC
         setb
                pushIndex.4
                                  ; keep even bank if SX28AC/18
ENDIF
IFDEF
         SX18
         setb
                pushIndex.4
ENDIF
         retp
```

Because the RAM in the SX52BD is stored in contiguous banks, and in the SX28AC the banks are separated by \$20, the IFDEF above will conditionally compile the setb instruction, allowing the pointer to memory to skip the non-existent banks in the SX28AC.

Macros help keep the source code clean. Example of a good macro for incrementing pointers to RAM:

```
; INCP/DECP macros for incrementing pointers to RAM
   INCP
         macro
                1
                             ; Increments a pointer to RAM
                \1
         inc
   IFNDEF SX48_52
          setb
                \1.4
                             ; If SX18 or SX28AC, keep bit 4 of the pointer =
   ENDIF
                             ; to jump from $1f to $30, etc.
endm
                             ; Decrements a pointer to RAM
DECP
         macro
         SX48_52
   IFDEF
          dec
                 \1
   ELSE
                \1.4
                             ; If SX18 or SX28AC, forces rollover to next bank
          clrb
                \1
                             ; if it rolls over (skips banks with bit 4 = 0)
          dec
          setb
                \1.4
                             ; Eg: $30 --> $20 --> $1f --> $1f
                             ; AND: $31 --> $21 --> $20 --> $30
   ENDIF
endm
```

Example of using a macro (INCP) to make the buffering source code easier to read:

BANK 0 Locations \$10 to \$1f

Because the SX48/52 can only access memory locations \$10 to \$1f directly, use these locations only as a last resort in programs written for the SX18/28, for compatibility with the SX48/52.

Extra RAM in the SX48/52

Because the SX18/28 has only half of the RAM of the SX48/52, avoid use of the extra RAM in programs written for the SX48/52, for compatibility with the SX18/28.

1.12 Lookup Tables (Jump Tables)

Routines should be clearly documented if they need to be completely within the first half of the page. Lookup tables that may be called by the programmer's own program should have protection against the table extending into the second half of a page. This can be done with the help of macros. By including a tableStart and tableEnd definition in the table, these macros will generate the error message "ERROR: Must be located in the first half of a page." if the table becomes misplaced.

```
; Error generating macros
    tableStart
           macro 0
                           ; Generates an error message if code that MUST be in
                           ; the first half of a page is moved into the second
    IF $ & $100
           ERROR 'Must be located in the first half of a page.'
    ENDIF
    ENDM
tableEnd
                           ; Generates an error message if code that MUST be in
           macro 0
                           ; the first half of a page is moved into the second
                           ; half.
    IF $ & $100
           ERROR 'Must be located in the first half of a page.'
    ENDIF
    ENDM
```

An assembler may implement this function in the future if these standard tableStart and tableEnd definitions are used.

This:

Becomes this:

```
jmp_table_1
; The code between the
; tableStart and tableEnd
; statements MUST be
; completely within the first
; half of a page. The routines
; it is jumping to must be in
; the same page as this table.
tableStart
     addpc,w
     jmp routine_1
      jmp routine_2
      jmp routine_3
     jmproutine_4
tableEnd
```

Note that the table must be in same page as the call to that table.

1.13 Memory Location Dependencies

Routines that are program memory location-dependent *must* be clearly marked:

If possible, defines or equates can be used to simplify this process:

```
mov m,#3 ; move upper nibble of address of strings into m
```

becomes

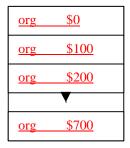
```
mov m,#STRINGS_ORG>>8; move upper nibble of address of strings into m
```

Now, as long as the STRINGS_ORG label precedes the strings, this subroutine will work properly, without regard to where the strings are located.

For routines with very location-specific data memory definitions, there should be ample documentation to indicate that the data memory cannot be moved around arbitrarily. Wherever possible, location-specific routines should be avoided.

1.14 Page Boundaries

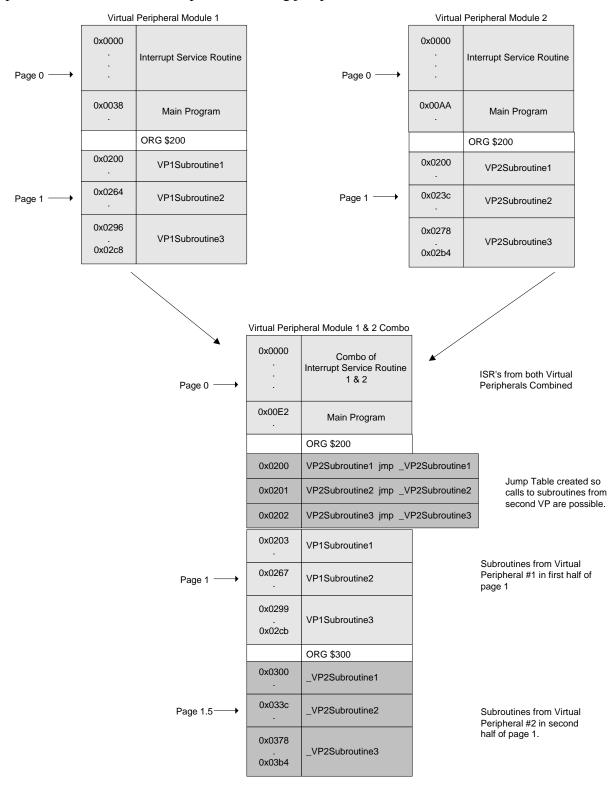
To ensure that several Virtual Peripheral modules, when pasted together, do not cross a page boundary without the programmer's knowledge, put an ORG statement with one instruction at every page boundary. This will generate an error if adding a subroutine moves another subroutine over a page boundary.



1.15 Subroutines in the Second Page of Program Memory

If two Virtual Peripheral modules are integrated together, and the subroutines for each Virtual Peripheral are to be placed into the same page, the callable subroutines from one of the Virtual Peripheral modules may need to be moved to the second half of a page. The problem this poses is that labels in the second half of a page can only be jumped to and not called. The solution is to create a jump table for the routines in the second half of the page. Unfortunately, the compromise is that calling a routine through a jump table adds a 3-cycle latency to the subroutine call, and therefore should only be used for routines that are not extremely speed-critical.

There is no simple way to make this job easier for the integrator, so the only solution is to provide ample documentation and examples on creating jump tables.



1.16 Defining ORG Statements

Place a table of ORG statements at the top of the source code with the starting addresses of all Virtual Peripheral modules. Use symbolic names for these addresses, rather than hard-coding them as literal values. This lends itself to separating the Virtual Peripheral modules into separate source files and creating a linker. Indicate whether each segment is moveable or not.

```
      UART_SUBS_ORG
      equ
      $300

      I2C_SUBS_ORG
      equ
      $400

      I2C_ISR_ORG
      equ
      $600
```

Now, instead of using the literal values, use the defined values:

```
org UART_SUBS
```

For smaller Virtual Peripheral modules, just use PAGE2_ORG, etc.

Example from included code:

```
; Program memory ORG defines
$0    ; Interrupt must always start at location zero
$100    ; Some more of the ISR is stored in location $100
INTERRUPT_ORG
                  equ $100 ; Some more of the ISR is stored in lo equ $1FB ; The program will jump here on reset. equ $200 ; The subroutines are always start at locat
                      equ
INTERRUPT_ORG2
RESET_ENTRY_ORG
SUBROUTINES ORG
                             $300 ; The strings are in location $300
STRINGS_ORG
                      equ
PAGE3_ORG
                               $400 ; Page 3 is empty
                      equ
MAIN_PROGRAM_ORG
                                $600 ; The main program is in the last page of program
                      equ
                                        ; memory.
```

And:

1.17 Main Program

Place the main routine at the end of the source code to make it easy to find. This means that if the first page is used for anything other than main program source code, a reset_entry must be placed in the first page, along with a page instruction and a jump instruction to the beginning of the main program.

Example:

Then, at the start of the main routine on another page:

1.18 Making Frequency Scalable

Whenever possible, Virtual Peripheral modules should be written so that they are scalable to work with virtually any interrupt rate. Virtual Peripheral modules that are written in this way include the A/D and D/A converters, the timers, FSK and DTMF generation and detection, LCD interface, and many others. Only the resolution or timing constants change as the interrupt rate changes.

Some Virtual Peripheral modules, however, require very specific interrupt rates. An example of this is the UART, which *must* be run at a 2ⁿ multiple of the desired UART rate. In this case, it must be made very clear to the programmer how to calculate the interrupt rates for all Virtual Peripheral modules, so that a frequency-dependent Virtual Peripheral like the UART can be chosen as the rate-determining factor, and all other Virtual Peripheral modules can have their constants re-calculated for the chosen rate.

```
; 19200 baud

; baud_bit = 4 ;for 19200 baud

; start_delay = 16+8+1 ;

; int_period = 163 ;
```

This type of definition gives the programmer no idea what to do if he wants to change the interrupt rate, while the following definition makes the change much more obvious:

```
Execution rate/16
             baud bit
;
                                                    ; For a baud rate of FS/16
                                      4
             start_delay
                                      16+8+1
      Execution rate/8
            baud_bit
                                                     ; For a baud rate of FS/8
                                      3
             start_delay
                                      8+4+1
      Execution rate/4
             baud_bit
                                                    ; For a baud rate of FS/4
                                      2
             start_delay
                                      4+2+1
```

This clears up what the definitions do to the UART speeds, and that the UART speed is tightly tied in to the interrupt rate. If the programmer wants a slower UART, or a slower interrupt rate, or both, the change is more intuitive.

If possible, let the compiler calculate the constant for the programmer, as in this example:

```
Fs = 9600 ; sampling frequency for DTMF detection ; 2^16 is the value of the phase accumulator

f697_1 equ ((Bits * 697)/Fs) & $0ff f697_h equ ((Bits * 697)/Fs) >> 8
```

In this example, the constant used to generate a 697 Hz signal is generated at compile time, as Fs changes. If the programmer wants to speed the execution rate of the frequency-generating Virtual Peripheral, he or she can simply change its execution rate in the ISR and scale the FS constant accordingly.

As long as the execution rates of each of the Virtual Peripheral modules are multiples of one another, they can all use the same retiw value. Unfortunately, the downfall of having only one jitter-free interrupt is that running Virtual Peripheral modules at rates that don't have a common denominator, such as 115 kHz and 250 kHz, is not possible.

1.19 Making Virtual Peripheral Modules Look and Act as Modules

- Keep each module *short*. The entire routine, beginning to end, should fit in the editor's window without scrolling.
- Indicate whether each ISR routine is variable-length or fixed-length in its header.
- Indicate the worst-case cycle time for each ISR routine in its header.
- Use only Scenix mnemonics as defined in the datasheet.
- Provide only *one* way in and out of each routine.
- Callable ISR routines: have a bank instruction for the routine performed before the call.
- Maximum ISR call-nesting: two levels, leaving nestable levels for the mainline program.
- As a rule, always return with a RETP instruction rather than a RET instruction (retp restores the page bits to the page of the call).

1.20 Define the ISR in Locations \$0 to \$1ff.

Keeps the ISR structure compact and easy to read. Use extra pages only if necessary.

1.21 Worst-Case ISR Cycle Count

The worst case ISR cycle count must not exceed RTCC interval. Exceeding the interrupt timing will cause the communication controller to miss an interrupt, which throws off the timing of every interrupt-driven Virtual Peripheral in the application.

1.22 Selecting the Interrupt Rate and Oscillator Frequency

Getting the desired interrupt rate from the desired oscillator frequency may be one of the most confusing parts of designing an interrupt-driven Virtual Peripheral. This procedure may be used to select these parameters.

- 1. Choose a desired interrupt frequency (irqFreq) based on the Virtual Peripheral modules you want to run.
 - Example: Choose 230.4 kHz for 4 times oversampling on a 57.6 kbps UART
- 2. Choose an oscillator frequency (oscFreq.) Higher sample-rate Virtual Peripheral modules will require higher oscillator frequencies. If power is not an issue for your design, a 50-MHz oscillator frequency is a safe bet for almost all Virtual Peripheral modules.

Example: Choose 50 MHz

- 3. Divide your oscillator frequency by your interrupt frequency. This is your ideal RETIW value.
 - Calculate RETIW value = (oscFreq/irqFreq)
 - Calculate 50 MHz/230.4 kHz = 217.01
- 4. Round your RETIW value to the nearest integer value between 0 and 255.
 - Round RETIW value to an integer
 - Round to 217
 - If the number exceeds 255, then slow down the RTCC by enabling its prescaler (reducing its time between RTCC increments by an integral power of 2 between 2 and 256), or choose a lower oscillator frequency. If the number is 90 or less, there may not be enough time to service each interrupt, so increase the oscillator frequency or decrease the interrupt frequency.
- 5. Calculate your actual interrupt frequency to see if it is close enough to your desired interrupt frequency by dividing the oscillator frequency by the RETIW value.
 - Actual Frequency = (oscFreq/RETIWVal * prescaler)
 - Check Actual Frequency = $50,000,000 \text{ Hz}/217 = 230.415 \text{ kHz} \cong 230.4 \text{ kHz}$
 - If the difference between the desired interrupt frequency and the actual interrupt frequency is too much, try recalculating with different oscillator frequencies.

1.23 Saving CPU Bandwidth

Instead of running the Virtual Peripheral on every interrupt, try to write it so it can run on every fourth or eighth interrupt. This makes integrating many Virtual Peripheral modules much less likely to overflow the number of cycles available for each interrupt, because the Interrupt Service Routine need only run one thread at a time.

1.24 Use the Multi-Threaded ISR Template

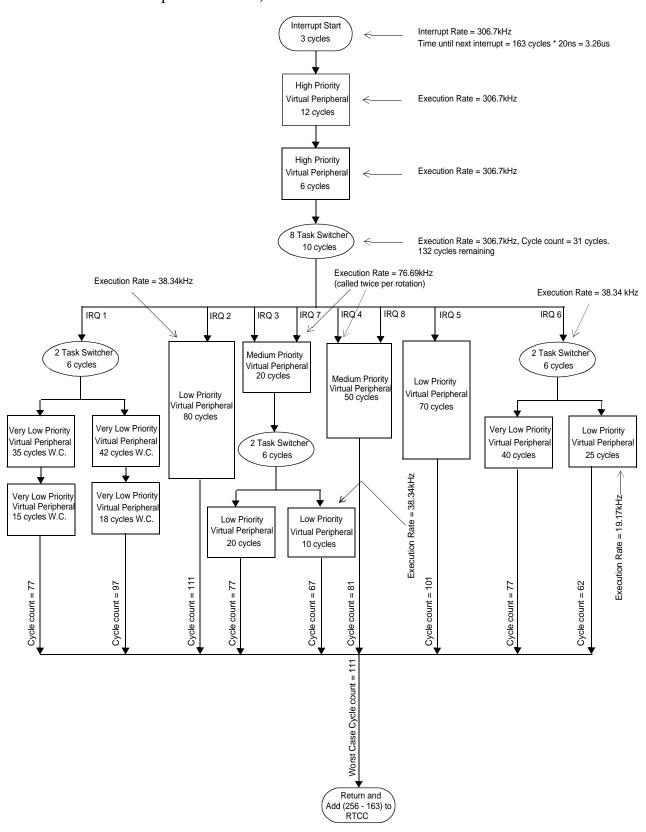
This method produces a *far* smaller worst-case cycle time count, and enables a larger number of Virtual Peripheral modules to run simultaneously. It also produces "empty" slots that future Virtual Peripheral modules can occupy.

- 1. Start with the multi-threaded ISR template shown on the next page.
- 2. Determine how often your tasks need to run. For example, a 9600-baud UART can run well at a sampling rate of only 38400 Hz, so don't run it faster than this.
- 3. Place your modules into the threads of the ISR. If a module needs to be run more often, call it at double the rate or at a rate that will meet its requirements.
- 4. Split complicated Virtual Peripheral modules into several modules, keeping the high-speed portions of the Virtual Peripheral modules as small and quick as possible, and run the more complicated, slower processing part of the Virtual Peripheral at a lower rate.

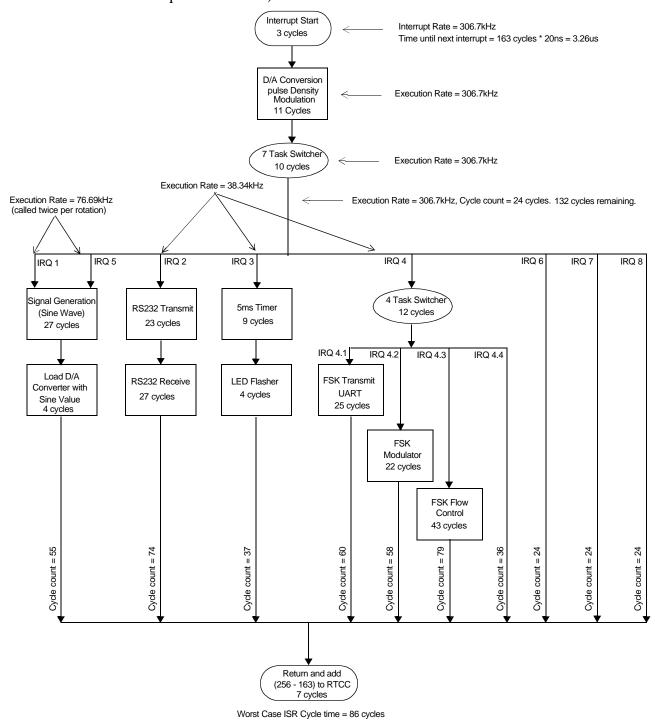
For example, in the Caller-ID detection program, the zero-cross-timer component of the Virtual Peripheral runs at double the speed of all of the other components, because it needs high resolution timing of the transitions on a pin. The other components of the Caller-ID Virtual Peripheral run at a slower rate, yet take a longer time to run when they are run. It is not necessary for them to run any faster, however, and doing so would increase the amount of CPU bandwidth used by the FSK detection Virtual Peripheral with no added benefit. (See the block diagram of the Caller-ID detection Interrupt Service Routine on the following pages.)

Because the ISR can now be viewed as a structure, made up of modules, it is easier for the user to increase and decrease the sampling rate by moving the modules around in the source code. Because only a few tasks are called in each interrupt, the flow of each interrupt is smaller and more easily understood. It is much simpler than an interrupt structure made up of many modules running consecutively, each jumping to the next.

Block diagram of a complex application, simplified through the use of a multi-threaded ISR (14 simultaneous Virtual Peripheral modules)

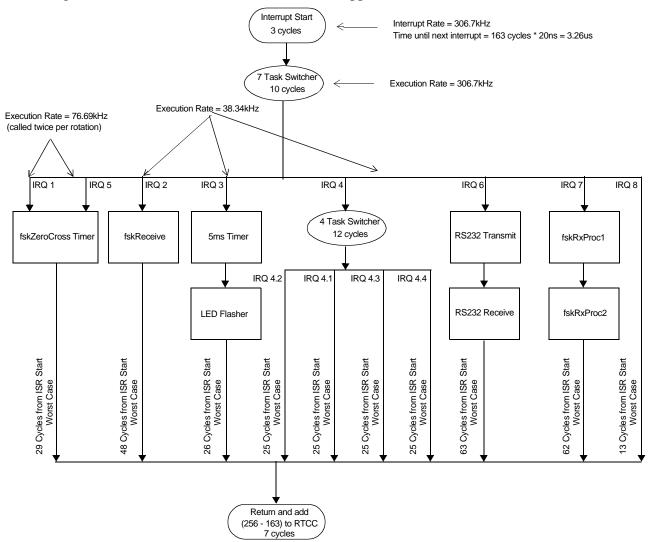


Block diagram of an actual application following this document's guidelines. (FSK generation with 10 simultaneous Virtual Peripheral modules)



Total Cycles allowed for ISR = 163

Block diagram of the ISR for the Caller-ID detection application.



Worst Case ISR Cycle time = 63 + 7 = 70 cycles Total Worst Case Cycles allowed for ISR = 163

SCENIX

Chapter 2

Source Code Template

2.1 Introduction

This chapter provides a Virtual Peripheral guideline compliant template. This template can be used to convert non-compliant Virtual Peripheral modules to compliant modules or develop new Virtual Peripheral modules.

2.2 Source Code Template

The template includes a skeleton Interrupt Service Routine (ISR) into which the Virtual Peripheral source code can be copied. It also provides the ISR multi-tasker which allows multiple threads of execution in the ISR. It includes initialization code, compiles and run, but the ISR and the main loop are empty.

```
; Copyright © [11/21/1999] Scenix Semiconductor, Inc. All rights reserved.
; Scenix Semiconductor, Inc. assumes no responsibility or liability for
; the use of this [product, application, software, any of these products].
; Scenix Semiconductor conveys no license, implicitly or otherwise, under
; any intellectual property rights.
; Information contained in this publication regarding (e.g.: application,
; implementation) and the like is intended through suggestion only and may
; be superseded by updates. Scenix Semiconductor makes no representation
; or warranties with respect to the accuracy or use of these information,
; or infringement of patents arising from such use or otherwise.
; Filename:
                  vp_guide_1_01.src
; Authors:
                  Chris Fogelklou
                  Applications Engineer
                  Scenix Semiconductor, Inc.
                  1.00
; Revision:
                  Put part datecode here.
; Part:
                  Put frequency here.
; Compiled using:
                Put assemblers/debuggers/hardware used here.
; Date Written: Jan 15, 2000
```

```
; Last Revised: Jan 15, 2000
; Program Description:
              Put program description here.
; Interface Pins:
              Put hardware interface pins here.
; Revision History:
              1.0 Put Revision History here.
; Target SX
; Uncomment one of the following lines to choose the SX18AC, SX20AC, SX28AC, SX48BD/ES,
; SX48BD, SX52BD/ES or SX52BD. For SX48BD/ES and SX52BD/ES, uncomment both defines,
; SX48_52 and SX48_52_ES.
; SX18_20
; SX28AC
SX48_52
; SX48_52_ES
; Assembler Used
; Uncomment the following line if using the Parallax SX-Key assembler. SASM assembler
; enabled by default.
;SX_Key
    ; Assembler directives:
         high speed external osc, turbo mode, 8-level stack, and extended option reg.
         SX18/20/28 - 4 pages of program memory and 8 banks of RAM enabled by default.
         SX48/52 - 8 pages of program memory and 16 banks of RAM enabled by default.
     IFDEF SX Kev
                        ;SX-Kev Directives
 IFDEF SX18_20
                        ;SX18AC or SX20AC device directives for SX-Key
              device SX18L,oschs2,turbo,stackx_optionx
 ENDIF
 IFDEF SX28AC
                        ;SX28AC device directives for SX-Key
              device SX28L, oschs2, turbo, stackx_optionx
 ENDIF
 IFDEF SX48_52_ES
                        ;SX48BD/ES or SX52BD/ES device directives for SX-Key
              device oschs, turbo, stackx, optionx
 ELSE
  IFDEF SX48_52
                        ;SX48/52/BD device directives for SX-Key
              device oschs2
  ENDIF
 ENDIF
                    50_000_000
              freq
ELSE
                        ;SASM Directives
```

```
IFDEF SX18_20
                         ;SX18AC or SX20AC device directives for SASM
              device SX18, oschs2, turbo, stackx, optionx
 ENDIF
 IFDEF SX28AC
                         ;SX28AC device directives for SASM
              device SX28AC, oschs2, turbo, stackx, optionx
 ENDIF
 IFDEF SX48_52_ES
                         ;SX48BD/ES or SX52BD/ES device directives for SASM
              device SX52BD, oschs, turbo, stackx, optionx
 ELSE
   IFDEF SX48_52
                         ;SX48BD or SX52BD device directives for SASM
              device SX52BD, oschs2
  ENDIF
 ENDIF
ENDIF
              id
                    'VPG '
              reset resetEntry ; set reset vector
; Macros
     ; Virtual Peripheral Guidelines Tip:
         To support compatibility between source code written for the SX28AC and the
SX52BD,
         use macros.
     ; Macro: _bank
     ; Sets the bank appropriately for all revisions of SX.
     ; This is required since the bank instruction has only a 3-bit operand, it cannot
     ; be used to access all 16 banks of the SX48/52. For this reason FSR.4 (for SX48/
     ; 52BD/ES)
     ; or FSR.7 (SX48/52bd production release) needs to be set appropriately, depending
     ; on the bank address being accessed. This macro fixes this.
     ; So, instead of using the bank instruction to switch between banks, use _bank
     ; instead.
     macro 1
bank
     bank \1
     IFDEF SX48_52
        IFDEF SX48_52_ES
           IF \1 & %00010000
                          ;SX48BD/ES and SX52BD/ES (engineering sample) bank
              setb
                    fsr.4
                          ; modifies FSR bits 5,6 and 7. FSR.4 needs to be set by
                           ;software.
           ENDIF
        ELSE
           IF \1 & %10000000
                          ;SX48BD and SX52BD (production release) bankinstruc-
tion
```

```
; modifies FSR bits 4,5 and 6. FSR.7 needs to be set by
               setb
                      fsr.7
            ELSE
               clrb
                     fsr.7
            ENDIF
         ENDIF
     ENDIF
     endm
     ; Macros for SX28AC/52 Compatibility
     ; Macro: _mode
     ; Sets the MODE register appropriately for all revisions of SX.
     ; This is required since the MODE (or MOV M,#) instruction has only a 4-bit operand.
     ; The SX18/20/28AC use only 4 bits of the MODE register, however the SX48/52BD have
     ; the added ability of reading or writing some of the MODE registers, and therefore use
     ; 5-bits of the MODE register. The MOV M,W instruction modifies all 8-bits of the
     ; MODE register, so this instruction must be used on the SX48/52BD to make sure the
     ; MODE
     ; register is written with the correct value. This macro fixes this.
     ; So, instead of using the MODE or MOV M, # instructions to load the M register, use
       _mode instead.
mode macro
     IFDEF SX48_52
 expand
          mov
               w, \# \setminus 1
                          ; loads the M register correctly for the SX48BD and SX52BD
          mov
               m,w
 noexpand
     ELSE
 expand
               m, \# \setminus 1
                          ; loads the M register correctly for the SX18AC,
          mov
                          ; and SX28AC
 noexpand
     ENDIF
     endm
     ; INCP/DECP macros for incrementing/decrementing pointers to RAM
     ; used to compensate for incompatibilities between SX28AC and SX52BD
     ; Virtual Peripheral Guidelines Tip:
           To support compatibility between source code written for the SX28AC and the
SX52BD,
           use macros. This macro compensates for the fact that RAM banks are contiquous
           in the SX52BD, but separated by 0x20 in the SX18/28.
     INCP
     macro
           1
           inc \1
     IFNDEF SX48_52
           setb \1.4
                          ; If SX18 or SX28AC, keep bit 4 of the pointer = 1
     ENDIF
                          ; to jump from $1f to $30, etc.
```

```
endm
DECP
     macro 1
     IFDEF SX48_52
          dec
               \1
     ELSE
                \1.4
                         ; If SX18 or SX28AC, forces rollover to next bank
          clrb
          dec
                \1
                          ; if it rolls over. (Skips banks with bit 4 = 0)
          setb
                \1.4
                          ; Eg: $30 --> $20 --> $1f --> $1f
     ENDIF
                          ; AND: $31 --> $21 --> $20 --> $30
endm
     ; Error generating macros
     ; Used to generate an error message if the label is unintentionally moved into the
     ; second half of a page. Use for lookup tables.
     ; Virtual Peripheral Guidelines Tip:
          Surround lookup tables with the tableStart and tableEnd macros. An error will
     ;
          be generated on assembly if the table crosses a page boundary.
     ;
     ;
          Example:
                lookupTable1
                     add
                          pc,w
                tableStart
                     retw
                          2.0
                     retw
                          -20
                     retw
                     retw
                          -40
                tableEnd
     tableStart
                           ; Generates an error message if code that MUST be in
        macro 0
                           ; the first half of a page is moved into the second half.
     if $ & $100
          ERROR 'Must be located in the first half of a page.'
     endif
endm
tableEnd
          macro 0
                          ; Generates an error message if code that MUST be in
                           ; the first half of a page is moved into the second half.
     if $ & $100
          ERROR 'Must be located in the first half of a page.'
endif
endm
; Data Memory address definitions
; These definitions ensure the proper address is used for banks 0 - 7 for 2K SX devices
; (SX18/20/28) and 4K SX devices (SX48/52).
IFDEF SX48_52
global_org
          =
                $0A
bank0_org
                $00
          =
bank1_org
                $10
```

bank2_org	=	\$20
bank3_org	=	\$30
bank4_org	=	\$40
bank5_org	=	\$50
bank6_org	=	\$60
bank7_org	=	\$70
ELSE		
alabal awa	_	\$08
global_org	=	
bank0_org	=	\$10
bank1_org	=	\$30
bank2_org	=	\$50
bank3_org	=	\$70
bank4_org	=	\$90
bank5_org	=	\$B0
bank6_org	=	\$D0
bank7_org	=	\$F0

ENDIF

```
; Global Register definitions
; NOTE: Global data memory starts at $0A on SX48/52 and $08 on SX18/20/28.
global_org
       orq
    ; Virtual Peripheral Guidelines Tip:
       Use only these defined label types for global registers. If an extra temporary
   ;
    ;
       register is required, adhere to these label types. For instance, if two
       temporary registers are required for the Interrupt Service Routine, use the
        label isrTemp1 for it.
    flags0
           global_org + 0
                       ; stores bit-wise operators like flags
       eau
                       ; and function-enabling bits (semaphores)
; VP: RS232 Receive
rs232RxFlag equ
           flags0.0
                       ; indicates the reception of a bit from the UART
flags1
       equ
           global_org + 1
                       ; stores bit-wise operators like flags
                       ; and function-enabling bits (semaphores)
localTemp0
        eau
           global_org + 2
                       ; temporary storage register
                       ; Used by first level of nesting
                       ; Never guaranteed to maintain data
localTemp1
                       ; temporary storage register
        equ
           global_org + 3
                       ; Used by second level of nesting
                       ; or when a routine needs more than one
                       ; temporary global register.
localTemp2
                       ; temporary storage register
        equ
           global_org + 4
                       ; Used by third level of nesting or by
                       ; main loop routines that need a loop
                       ; counter, etc.
isrTemp0
           global_org + 5
                       ; Interrupt Service Routine's temp register.
        eau
                       ; Don't use this register in the mainline.
; RAM Bank Register definitions
; Bank 0
    org
            bank0_org
bank0
            Ś
    ; Virtual Peripheral Guidelines Tip:
        - Avoid using bank0 in programs written for SX48/52.
    ; Bank 1
    bank1_org
       ora
```

```
; Virtual Peripheral Guidelines Tip:
           Tip 1:
           Indicate which Virtual Peripherals a portion of source code or declaration be
           longs to with a ;VP: VirtualPeripheralName comment.
           Tip 2:
           All RAM location declaration names should be
           - left justified
           - less than 2 tabs in length
           - written in hungarian notation
           - prefixed by a truncated version of the Virtual Peripheral's name
           Examples:
           ; VP: RS232 Transmit
           rs232TxBank
                                        $
                                              ;RS232 Transmit bank
           rs232TxHigh
                            ds
                                        1
                                              ;hi byte to transmit
           rs232TxLow
                                        1
                                              ;low byte to transmit
                            ds
           rs232TxCount
                            ds
                                        1
                                              ;number of bits sent
     ;
           rs232TxDivide
                            ds
                                        1
                                              ;xmit timing (/16) counter
           rs232TxString
                            ds
                                        1
                                              ;the address of the string to be sent
           rs232TxByte
                            ds
                                        1
                                              ;semi-temporary serial register
     ; VP: ISR Multithreader
isrMultiplex
                 ds
                            ; The isrMultiplex register is used to switch to a new
                             ; execution thread on each pass of the ISR.
      ; Bank 2
                 org
                        bank2_org
bank2
     ; Bank 3
                 org
                       bank3_org
bank3
      ; Bank 4
                 org
                       bank4_org
bank4
                       $
                 org
                       bank5_org
bank5
                 =
                       $
```

```
bank6_org
             org
bank6
                  $
                   bank7_org
             ora
bank7
                  $
IFDEF SX48_52
    ; * * * * * * *
    ; Bank 8
                                ;bank 8 address on SX52BD
             org
                  $80
bank8
                  $
    ; Virtual Peripheral Guidelines Tip:
         - This extra memory is not available in the SX18/28, so don't use it for Virtual
          Peripherals written for both platforms.
    $90
                               ;bank 9 address on SX52BD
             org
bank9
                  $
    ;bank A address on SX52BD
                  $A0
             org
bankA
                  $
    ; Bank B
                  $B0
                               ;bank B address on SX52BD
             org
bankB
    ; Bank C
                  $C0
                               ;bank C address on SX52BD
             org
bankC
                  $
```

```
$D0
                                   ;bank D address on SX52BD
               org
bankD
                    $
     ; Bank E
                   $E0
                                   ;bank E address on SX52BD
               org
bankE
                    $
     ; Bank F
                   $F0
                                   ;bank F address on SX52BD
               orq
bankF
                   $
ENDIF
     ; Pin Definitions:
     ; Virtual Peripheral Guidelines Tip:
          - Store all initialization constants for the I/O in the same area, so
           pins can be easily moved around.
          - Pin definitions should follow the same format guidelines as RAM definitions
               - Left justified
               - Hungarian Notation
               - Less that 2 tabs in length
               - Indicate the Virtual Peripheral the pin is used for
          - Only use symbolic names to access a pin/port in the source code.
          - Example:
               ; VP: RS232 Transmit
                   rs232TxPin equ
                                   ra.3
     RA_latch
         equ
               %00000000
                              ;SX18/20/28/48/52 port A latch init
RA_DDIR
               %11111111
                              ;SX18/20/28/48/52 port A DDIR value
          equ
RA_LVL
          equ
               %0000000
                             ;SX18/20/28/48/52 port A LVL value
               %00000000
                             ;SX18/20/28/48/52 port A PLP value
RA_PLP
         equ
RB_latch
               %0000000
                              ;SX18/20/28/48/52 port B latch init
         equ
RB_DDIR
               %11111111
                             ;SX18/20/28/48/52 port B DDIR value
         equ
RB_ST
          equ
               %11111111
                              ;SX18/20/28/48/52 port B ST value
RB_LVL
         equ
               %0000000
                              ;SX18/20/28/48/52 port B LVL value
RB_PLP
         equ
               %0000000
                             ;SX18/20/28/48/52 port B PLP value
               %0000000
                              ;SX18/20/28/48/52 port C latch init
RC_latch
         equ
RC_DDIR
               %11111111
                              ;SX18/20/28/48/52 port C DDIR value
          equ
```

```
RC_ST
               %11111111
                                ;SX18/20/28/48/52 port C ST value
          equ
                %00000000
RC_LVL
          equ
                                ;SX18/20/28/48/52 port C LVL value
                %00000000
                                ;SX18/20/28/48/52 port C PLP value
RC_PLP
          equ
IFDEF SX48_52
                                ;SX48BD/52BD Port initialization values
RD_latch
                800000000
                                ;SX48/52 port D latch init
          equ
RD_DDIR
                                ;SX48/52 port D DDIR value
          equ
                %11111111
RD_ST
          equ
                %11111111
                                ;SX48/52 port D ST value
RD_LVL
          equ
                %00000000
                                ;SX48/52 port D LVL value
RD_PLP
                               ;SX48/52 port D PLP value
          equ
                800000000
               %00000000
RE_latch
                               ;SX48/52 port E latch init
          equ
                               ;SX48/52 port E DDIR value
RE_DDIR
                %11111111
          equ
RE_ST
          equ
                %11111111
                               ;SX48/52 port E ST value
                %00000000
                                ;SX48/52 port E LVL value
RE_LVL
          equ
                800000000
                                ;SX48/52 port E PLP value
RE_PLP
          equ
ENDIF
; Program constants
; Virtual Peripheral Guidelines Tip:
          To calculate the interrupt period in cycles:
          - First, choose the desired interrupt frequency
                - Should be a multiple of each Virtual Peripherals sampling frequency.
                - Example: 19200kHz UART sampling rate * 16 = 307.200kHz
          - Next, choose the desired oscillator frequency.
                - 50MHz, for example.
          - Perform the calculation int_period = (osc. frequency / interrupt frequency)
                                     = (50MHz / 307.2kHz)
                                     = 162.7604
          - Round int_period to the nearest integer:
          - Now calculate your actual interrupt rate:
                                     = osc. frequency / int_period
                                     = 50MHz / 163
                                     = 306.748 \text{kHz}
          - This interrupt frequency will be the timebase for all of the Virtual
            Peripherals
     ; Gives an interrupt period at 50MHz of (217 * (1/50000000)s) =
int_period
                217
                     ; 4.34us
                     ; Which gives an interrupt frequency of (1/4.34us)Hz = 230414kHz
     ; Virtual Peripheral Guidelines Tip:
          - Include all calculations for Virtual Peripheral constants for any sample
            rate.
          - Relate all Virtual Peripheral constants to the sample rate of the Virtual
            Peripheral.
          - Example:
                ; VP: 5ms Timer
               TIMER_DIV_CONSTequ 192; This constant = timer sample rate/200Hz = 192
```

```
IFDEF SX48_52
     ; SX48BD/52BD Mode addresses
     ; *On SX48BD/52BD, most registers addressed via mode are read and write, with the
     ; exception of CMP and WKPND which do an exchange with W.
     ; Timer (read) addresses
TCPL_R
           equ
                 $00
                            ;Read Timer Capture register low byte
TCPH_R
           equ
                 $01
                            ;Read Timer Capture register high byte
                           ;Read Timer R2 low byte
TR2CML_R
                 $02
         equ
TR2CMH_R
                 $03
                           ;Read Timer R2 high byte
           equ
                 $04
                           Read Timer R1 low byte
TR1CML_R
           equ
TR1CMH_R
                 $05
                           ;Read Timer R1 high byte
           equ
                 $06
                           ;Read Timer control register B
TCNTB_R
           equ
TCNTA_R
                 $07
                           ;Read Timer control register A
           equ
; Exchange addresses
                            ; Exchange Comparator enable/status register with W
CMP
           equ
                 $08
WKPND
           equ
                 $09
                            ; Exchange MIWU/RB Interrupts pending with W
; Port setup (read) addresses
WKED R
                 $0A
                            ;Read MIWU/RB Interrupt edge setup, 0 = falling, 1 = rising
        equ
                            ;Read MIWU/RB Interrupt edge setup,
WKEN_R
          equ
                 $0B
                            ;0 = enabled, 1 = disabled
ST_R
           equ
                 $0C
                            ;Read Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
                 $0D
                            ;Read Port Level setup, 0 = CMOS, 1 = TTL
LVL_R
           equ
PLP_R
                 $0E
                            ;Read Port Weak Pullup setup, 0 = enabled, 1 = disabled
           equ
DDIR_R
                 $0F
                            Read Port Direction
           equ
; Timer (write) addresses
TR2CML_W
        equ
               $12
                            ;Write Timer R2 low byte
                            ;Write Timer R2 high byte
TR2CMH_W
           equ
                 $13
TR1CML_W
                $14
                            ;Write Timer R1 low byte
           equ
                           ;Write Timer R1 high byte
TR1CMH_W
                 $15
           equ
TCNTB_W
                 $16
                           ;Write Timer control register B
           equ
                            ;Write Timer control register A
TCNTA_W
           equ
                 $17
; Port setup (write) addresses
WKED_W
           eau
                 $1A
                            ;Write MIWU/RB Interrupt edge setup,
                            ;0 = falling, 1 = rising
                 $1B
WKEN_W
                            ;Write MIWU/RB Interrupt edge setup,
           equ
                            ;0 = enabled, 0 = enabled,1 = disabled
ST_W
           equ
                 $1C
                            ;Write Port Schmitt Trigger setup,
                            ;0 = enabled, 1 = disabled
LVL_W
           equ
                 $1D
                            ;Write Port Level setup, 0 = CMOS, 1 = TTL
                            ;Write Port Weak Pullup setup, 0 = enabled, 1 = disabled
PLP_W
                 $1E
           equ
DDIR_W
                 $1F
                            ;Write Port Direction
           equ
ELSE
; SX18AC/20AC/28AC Mode addresses
; *On SX18/20/28, all registers addressed via mode are write only, with the exception of
; CMP and WKPND which do an exchange with W.
; Exchange addresses
CMP
                 $08
                            ; Exchange Comparator enable/status register with W
           eau
WKPND
                 $09
                            ;Exchange MIWU/RB Interrupts pending with W
           eau
```

```
; Port setup (read) addresses
WKED_W
         equ
               $0A
                          ;Write MIWU/RB Interrupt edge setup,
                          ;0 = falling, 1 = rising
WKEN_W
                          ;Write MIWU/RB Interrupt edge setup,
               $0B
          equ
                          ;0 = enabled, 1 = disabled
ST_W
               SOC
                         ;Write Port Schmitt Trigger setup,
          equ
                          ;0 = enabled, 1 = disabled
LVL_W
          equ
               $0D
                          ;Write Port Schmitt Trigger setup,
                          ;0 = enabled, 1 = disabled
PLP_W
          equ
               $0E
                          ;Write Port Schmitt Trigger setup,
                          ;0 = enabled, 1 = disabled
DDIR_W
               $0F
                          ;Write Port Direction
          eau
ENDIF
; Program memory ORG defines
     ; Virtual Peripheral Guidelines Tip:
          - Place a table at the top of the source with the starting addresses of all of
           the components of the program.
     INTERRUPT_ORG
               equ
                     $0
                         ; Interrupt must always start at location zero
RESET_ENTRY_ORG
                    $1FB ; The program will jump here on reset.
               equ
                     $200 ; The subroutines are in this location
SUBROUTINES_ORG
               equ
                     $300 ; The strings are in location $300
STRINGS_ORG
               equ
                    $400 ; Page 3 is empty
PAGE3_ORG
               equ
                    $600 ; The main program is in the last page of program memory.
MAIN_PROGRAM_ORG
               equ
```

```
INTERRUPT_ORG
                     ; First location in program memory.
:______
; Interrupt Service Routine
:______
; Note: The interrupt code must always originate at address $0.
; Interrupt Frequency = (Cycle Frequency / -(retiw value)) For example:
; With a retiw value of -163 and an oscillator frequency of 50MHz, this
; code runs every 3.26us.
;______
                 ;3 The interrupt service routine...
;-----
; VP: VP Multitasker
    ; Virtual Peripheral Guidelines Tip:
    ; - Multi-thread the Interrupt Service Routine
         - Produces a FAR smaller worst-case cycle time count, and enables a larger
         number of VP's to run simultaneously. Also produces "empty" slots that future
         VP's can be copied and pasted into easily.
         - Determine how often your tasks need to run. (9600bps UART can run well at a
          sampling rate of only 38400Hz, so don't run it faster than this.)
         - Strategically place each "module" into the threads of the ISR. If a module
         must be run more often, just call it's module at double the rate or quadruple
         the rate, etc....
         - Split complicated Virtual Peripherals into several modules, keeping the
         high-speed portions of the Virtual Peripherals as small and quick as possible,
         and run the more complicated, slower processing part of the Virtual Peripheral
          at a lower rate.
    ;-----
    ; Virtual Peripheral Multitasker: up to 24 individual threads, each running at
                           the interrupt rate/24. Change the
         Input variable(s): isr_multiplex: variable used to choose threads
         Output variable(s): None, executes the next thread
         Variable(s) affected: isr_multiplex
         Flag(s) affected: None
         Program Cycles: 9 cycles (turbo mode)
    ;-----
             isrMultiplex
    _bank
                          ; 1
                          ;1
             isrMultiplex
    inc
                              ; toggle interrupt rates
             w,isrMultiplex
                          ;1
    mov
; The code between the tableBegin and tableEnd statements MUST be
; completely within the first half of a page. The routines
; it is jumping to must be in the same page as this table.
tableStart
                                ; Start all tables with this macro.
    jmp
             pc+w
                           ; 3
    jmp
             isrThread1
                          ;3,9 cycles.
    jmp
             isrThread2
    jmp
             isrThread3
    jmp
             isrThread4
             isrThread1
    ami
             isrThread5
    jmp
             isrThread6
    jmp
```

```
isrThread7
   jmp
                      ;
   jmp
          isrThread1
           isrThread8
   jmp
           isrThread9
   jmp
           isrThread10
   jmp
           isrThread1
   jmp
           isrThread11
   jmp
   jmp
           isrThread12
   jmp
           isrThread13
tableEnd
                      ; End all tables with this macro.
;-----
; VP: VP Multitasker
; ISR TASKS
:______
                  ; Serviced at ISR rate / 4
isrThread1
:_____
   ; Virtual Peripheral Guidelines Tip:
       The sample rate of this section of code is the isr rate / 4, because it is jumped
       to in every 4th entry in the VP Multitaskers table. To increase the
       sample rate, put more calls to this thread in the Multitasker's jump table.
   ;7 cycles until mainline program resumes execution
isrThread2
                  ; Serviced at ISR rate / 16
      isrOut
   jmp
                  ;7 cycles until mainline program resumes execution
isrThread3
                  ; Serviced at ISR rate / 16
;-----
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
;______
isrThread4
                  ; Serviced at ISR rate / 16
;-----
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
:______
                  ; Serviced at ISR rate / 16
isrThread5
;-----
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
:______
isrThread6
                  ; Serviced at ISR rate / 16
;______
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
isrThread7
                   ; Serviced at ISR rate / 16
   jmp
      isrOut
                  ;7 cycles until mainline program resumes execution
;-----
isrThread8
                  ; Serviced at ISR rate / 16
```

```
:-----
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
isrThread9
                  ; Serviced at ISR rate / 16
      isrOut
                  ;7 cycles until mainline program resumes execution
   jmp
isrThread10
                  ; Serviced at ISR rate / 16
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
;______
isrThread11
                  ; Serviced at ISR rate / 16
;-----
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
:_____
isrThread12
                  ; Serviced at ISR rate / 16
;-----
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
isrThread13
                  ; Serviced at ISR rate / 16
                  ; This thread must reload the isrMultiplex register
                  ; since it is the last one to run in a rotation.
   bank isrMultiplex
   mov isrMultiplex, #255 ; Reload isrMultiplex so isrThread1 will be run
                  ; on next interrupt.
   jmp isrOut
            _____
;-----
   ______
   mov w,#-int_period ;1
                     ; return and add -int_period to the RTCC
                 ;3 ; using the retiw instruction.
retiw
:______
RESET_ENTRY_ORG
; Virtual Peripheral Guidelines Tip:
       The main program operation should be easy to find, so place it at the end of the
       program code. This means that if the first page is used for anything other than
       main program source code, a reset_entry must be placed in the first page, along
       with a 'page' instruction and a 'jump' instruction to the beginning of the
       main program.
   ; Program starts here on power-up
resetEntry
   page _resetEntry
```

```
jmp
      _resetEntry
   ; Virtual Peripheral Guidelines Tip:
      ORG statements should use predefined labels rather than literal values.
   org
   SUBROUTINES ORG
; Subroutines
STRINGS_ORG ; This label defines where strings are kept in program space.
; Put String Data Here
;______
; Example:
           13,10,'UART Demo',0
;_hello
       dw
   ; Virtual Peripheral Guidelines Tip:
      - Routines that use location-dependant data, such as in example below, should
       use a LABEL rather than a literal value as their input. Example:
       instead of
                   ; move upper nybble of address of strings into m
         mov
           m,#3
       1150
         mov
            m, #STRINGS_ORG>>8; move upper nybble of address of strings into m
   org
  PAGE3 ORG
; Virtual Peripheral Guidelines Tip:
      To ensure that several Virtual Peripherals, when pasted together, do not cross
      a page boundary without the integrator's knowledge, put an ORG statement and
      one instruction at every page boundary. This will generate an error if a pasted
      subroutine moves another subroutine to a page boundary.
   ; This instruction will cause an assembler error if the source code
   ami
         ; before the org statement inadvertantly crosses a page boundary.
MAIN_PROGRAM_ORG
```

```
; RESET VECTOR
       ; Program execution begins here on power-up or after a reset
_resetEntry
       ; Initialize all port configuration
             _mode ST_W
                                         ;point MODE to write ST register
                                         ;Setup RB Schmitt Trigger, 0 = enabled, 1 = disabled
             mov
                    w, #RB_ST
                    !rb,w
             mov
                                         ;Setup RC Schmitt Trigger, 0 = enabled, 1 = disabled
                    w, #RC_ST
             mov
             mov
                    !rc,w
IFDEF SX48_52
             mov
                    w, #RD_ST
                                         ;Setup RD Schmitt Trigger, 0 = enabled, 1 = disabled
             mov
                    !rd,w
             mov
                    w, #RE_ST
                                         ;Setup RE Schmitt Trigger, 0 = enabled, 1 = disabled
             mov
                    !re,w
ENDIF
             _mode LVL_W
                                         ;point MODE to write LVL register
             mov
                    w,#RA_LVL
                                         ;Setup RA CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
                    !ra,w
                    w, #RB_LVL
                                         ;Setup RB CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
             mov
                    !rb.w
                    w,#RC_LVL
                                         ;Setup RC CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
             mov
                     !rc,w
IFDEF SX48_52
                    w, #RD_LVL
                                         ;Setup RD CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
             mov
                    !rd,w
                    w, #RE_LVL
                                         ;Setup RE CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
                    !re,w
             mov
ENDIF
                    PLP_W
                                         ;point MODE to write PLP register
             mode
             mov
                    w, #RA_PLP
                                         ;Setup RA Weak Pull-up, 0 = enabled, 1 = disabled
             mov
                    !ra,w
                                         ;Setup RB Weak Pull-up, 0 = enabled, 1 = disabled
                    w, #RB_PLP
             mov
             mov
                    !rb,w
                                         ;Setup RC Weak Pull-up, 0 = enabled, 1 = disabled
             mov
                    w, #RC_PLP
             mov
                     !rc,w
IFDEF SX48_52
             mov
                    w, #RD_PLP
                                         ;Setup RD Weak Pull-up, 0 = enabled, 1 = disabled
                    !rd.w
             mov
                    w, #RE_PLP
                                         ;Setup RE Weak Pull-up, 0 = enabled, 1 = disabled
             mov
             mov
                    !re,w
ENDIF
                                          ;point MODE to write DDIR register
             _mode DDIR_W
                    w, #RA_DDIR
                                         ;Setup RA Direction register, 0 = output, 1 = input
             mov
                    !ra,w
             mov
                    w, #RB_DDIR
                                         ;Setup RB Direction register, 0 = output, 1 = input
             mov
             mov
                    !rb,w
             mov
                    w, #RC_DDIR
                                         ;Setup RC Direction register, 0 = output, 1 = input
             mov
                    !rc,w
IFDEF SX48_52
                    w, #RD_DDIR
                                         ;Setup RD Direction register, 0 = output, 1 = input
             mov
             mov
                    !rd.w
```

```
w, #RE_DDIR
                                         ;Setup RE Direction register, 0 = output, 1 = input
             mov
             mov
                    !re,w
ENDIF
                                         ;Initialize RA data latch
             mov
                    w,#RA_latch
                    ra,w
             mov
                    w,#RB_latch
                                        ;Initialize RB data latch
             mov
             mov
                    rb,w
             mov
                    w,#RC_latch
                                        ;Initialize RC data latch
             mov
                    rc,w
IFDEF SX48_52
                                        ;Initialize RD data latch
             mov
                    w,#RD_latch
                    rd,w
             mov
                                        ;Initialize RE data latch
                    w,#RE_latch
             mov
             mov
                    re,w
ENDIF
       ; Clear all Data RAM locations
zeroRam
IFDEF SX48_52
                                         ;SX48/52 RAM clear routine
             mov
                    w,#$0a
                                         ;reset all ram starting at $0A
             mov
                    fsr,w
                                         ; clear using indirect addressing
:zeroRam
             clr
                    ind
             incsz fsr
                                         ;repeat until done
             jmp
                    :zeroRam
                                        ;clear bank 0 registers
             _bank bank0
             clr
                    $10
             clr
                    $11
             clr
                    $12
             clr
                    $13
             clr
                    $14
             clr
                    $15
             clr
                    $16
             clr
                    $17
             clr
                    $18
             clr
                    $19
             clr
                    $1a
             clr
                    $1b
             clr
                    $1c
             clr
                    $1d
             clr
                    $1e
             clr
                    $1f
ELSE
                                         ;SX18/20/28 RAM clear routine
                    fsr
                                         ;reset all ram banks
             clr
:zeroRam
             sb
                    fsr.4
                                         ; are we on low half of bank?
             setb fsr.3
                                        ; If so, don't touch regs 0-7
             clr
                    ind
                                         ; clear using indirect addressing
             incsz fsr
                                         ;repeat until done
             jmp
                    :zeroRam
ENDIF
       ; Initialize program/VP registers
```

```
; Setup and enable RTCC interrupt, WREG register, RTCC/WDT prescaler
      ; Virtual Peripheral Guidelines Tip:
            The suggested default values for the option register are:
            - Bit 7 set to 0: location $01 addresses the W register (WREG
            - Bit 5 set to 1: RTCC increments on internal transitions
            - Bit 3 set to 1: Prescaler assigned to WatchDog Timer
            If a routine must change the value of the option register (for example, to
            access the RTCC register directly), then it should restore the default value
            for the option register before exiting.
      RTCC_ON
                  %10000000
                                    ; Enables RTCC at address $01 (RTW hi)
                                    ; *WREG at address $01 (RTW lo) by default
RTCC_ID
                  %01000000
                                    ;Disables RTCC edge interrupt (RTE_IE hi)
                                    ;*RTCC edge interrupt (RTE_IE lo) enabled by
                                    ;default
RTCC_INC_EXT =
                  %00100000
                                    ; Sets RTCC increment on RTCC pin transition (RTS hi)
                                    ;*RTCC increment on internal instruction (RTS lo)
                                    is default;
RTCC_FE
                  %00010000
                                    ;Sets RTCC to increment on falling edge (RTE_ES hi)
                                    ;*RTCC to increment on rising edge (RTE_ES lo) is
                                    ;default
RTCC_PS_ON
                 %00000000
                                    ; Assigns prescaler to RTCC (PSA lo)
            =
RTCC_PS_OFF =
                 %00001000
                                   ;Assigns prescaler to WDT (PSA lo)
PS_000 =
                 %00000000
                                   ;RTCC = 1:2, WDT = 1:1
                                   RTCC = 1:4, WDT = 1:2
PS_001
                 %00000001
           =
PS_010
                %00000010
                                   RTCC = 1:8, WDT = 1:4
PS_011
                %00000011
                                   ;RTCC = 1:16, WDT = 1:8
PS_100
          =
                %00000100
                                   ;RTCC = 1:32, WDT = 1:16
PS_101
          =
                 %00000101
                                   RTCC = 1:64, WDT = 1:32
PS_110
                                   ;RTCC = 1:128, WDT = 1:64
                 %00000110
           =
PS_111
                                   ;RTCC = 1:256, WDT = 1:128
                  %00000111
                 RTCC_PS_OFF | PS_111 ; the default option setup for this program.
OPTIONSETUP
           equ
                                ; setup option register for RTCC interrupts enabled
            mov
                 w,#OPTIONSETUP
            mov
                  !option,w
                                   ; and no prescaler.
                  @mainLoop
            jmp
; MAIN PROGRAM CODE
mainLoop
            jmp
                  mainLoop
```

; * * * * * * * * * * *	************
END	;End of program code
; * * * * * * * * * * *	***********************
; * * * * * * * * * * *	************
,	************
; * * * * * * * * * * *	*************

; * * * * * * * * * * *	************



Chapter 3

Adding A Virtual Peripheral to the **Source Code Template**

3.1 Introduction

This chapter shows the interaction between the Virtual Peripheral guideline compliant template and a Virtual Peripheral module.

3.2 Source Code Template With A Virtual Peripheral Example

The following source code shows how a UART Virtual Peripheral module can be added to the source code template described.

```
; Copyright @ [11/21/1999] Scenix Semiconductor, Inc. All rights reserved.
; Scenix Semiconductor, Inc. assumes no responsibility or liability for
; the use of this [product, application, software, any of these products].
; Scenix Semiconductor conveys no license, implicitly or otherwise, under
; any intellectual property rights.
; Information contained in this publication regarding (e.g.: application,
; implementation) and the like is intended through suggestion only and may
; be superseded by updates. Scenix Semiconductor makes no representation
; or warranties with respect to the accuracy or use of these information,
; or infringement of patents arising from such use or otherwise.
; Filename:
                   vpg_UART_1_0.src
; Authors:
                    Chris Fogelklou
                    Applications Engineer
                    Scenix Semiconductor, Inc.
                    1.00
; Revision:
; Part:
                    Put part datecode here.
                    25MHz
; Frea:
; Compiled using: Put assemblers/debuggers/hardware used here.
; Date Written: Jan 15, 2000
; Last Revised: Jan 15, 2000
```

```
; Program Description:
               Virtual Peripherals Guidelines:
               Example source code, running at 25MHz, with just a transmit
               and receive UART. The code implements a 9600bps UART in software,
               and the speed is defineable through equate statements.
; Interface Pins:
               rs232RxPin
                            equ
                                ra.2
                                         ;UART receive input
               rs232TxPin
                           equ
                                  ra.3
                                         ;UART transmit output
; Revision History:
               1.0 Used the VP Guidelines multi-threaded example and inserted a UART
                     for an example of code that actually works.
               Put rest of revision history here...
; Target SX
; Uncomment one of the following lines to choose the SX18AC, SX20AC, SX28AC, SX48BD/ES,
; SX48BD, SX52BD/ES or SX52BD. For SX48BD/ES and SX52BD/ES, uncomment both defines,
; SX48\_52 and SX48\_52\_ES.
;SX18_20
SX28
;SX48_52
;SX48_52_ES
; Assembler Used
; Uncomment the following line if using the Parallax SX-Key assembler. SASM assembler
; enabled by default.
;SX_Key
     ; Assembler directives:
          high speed external osc, turbo mode, 8-level stack, and extended option reg.
          SX18/20/28 - 4 pages of program memory and 8 banks of RAM enabled by default.
          SX48/52 - 8 pages of program memory and 16 banks of RAM enabled by default.
     IFDEF SX_Key
                               ;SX-Key Directives
 IFDEF SX18_20
                               ;SX18AC or SX20AC device directives for SX-Key
               device SX18L,oschs2,turbo,stackx_optionx
 ENDIF
 IFDEF SX28
                               ;SX28AC device directives for SX-Key
               device SX28L, oschs2, turbo, stackx_optionx
 ENDIF
 IFDEF SX48_52_ES
                               ;SX48BD/ES or SX52BD/ES device directives for
                               ;SX-Key
               device oschs, turbo, stackx, optionx
   IFDEF SX48_52
                               ;SX48/52/BD device directives for SX-Key
               device oschs2
```

```
ENDIF
 ENDIF
                      25_000_000
                freq
ELSE
                                 ;SASM Directives
 IFDEF SX18_20
                                  ;SX18AC or SX20AC device directives for SASM
                device SX18, oschs2, turbo, stackx, optionx
 ENDIF
 IFDEF SX28
                                  ;SX28AC device directives for SASM
                device SX28, oschs2, turbo, stackx, optionx
 ENDIF
                                  ;SX48BD/ES or SX52BD/ES device directives for SASM
 IFDEF SX48_52_ES
                device SX52, oschs, turbo, stackx, optionx
 ELSE
   IFDEF SX48_52
                                  ;SX48BD or SX52BD device directives for SASM
                device SX52, oschs2
   ENDIF
 ENDIF
ENDIF
                id
                      'VPGUART1'
                reset resetEntry
                                 ; set reset vector
; Macros
     ; Virtual Peripheral Guidelines Tip:
           To support compatibility between source code written for the SX28 and the SX52,
           use macros.
     ; Macro: _bank
     ; Sets the bank appropriately for all revisions of SX.
     ; This is required since the bank instruction has only a 3-bit operand, it cannot
     ; be used to access all 16 banks of the SX48/52. For this reason FSR.4
     ; (for SX48/52BD/ES)or FSR.7 (SX48/52bd production release) needs to be set
     ; appropriately, depending on the bank address being accessed. This macro fixes this.
     ; So, instead of using the bank instruction to switch between banks, use _bank
     ; instead.
     bank macro 1
     bank
          \1
     IFDEF SX48_52
           IFDEF SX48_52_ES
            IF \1 & %00010000
                                 ;SX48BD/ES and SX52BD/ES (engineering sample) bank
                                  ;instruction
                                  ; modifies FSR bits 5,6 and 7. FSR.4 needs to be set
                setb
                     fsr.4
                                 ; by software.
            ENDIF
     ELSE
           IF \1 & %10000000
                                 ;SX48BD and SX52BD (production release) bank
                            ;instruction
                      fsr.7
                                  ; modifies FSR bits 4,5 and 6. FSR.7 needs to be set
                setb
                                  ; by software.
```

```
ELSE
                    clrb fsr.7
      ENDIF
  ENDIF
ENDIF
endm
```

```
; Macros for SX28/52 Compatibility
     ; Macro: _mode
     ; Sets the MODE register appropriately for all revisions of SX.
     ; This is required since the MODE (or MOV M,#) instruction has only a 4-bit operand.
     ; The SX18/20/28AC use only 4 bits of the MODE register, however the SX48/52BD have
     ; the added ability of reading or writing some of the MODE registers, and therefore use
     ; 5-bits of the MODE register. The MOV M,W instruction modifies all 8-bits of the
     ; MODE register, so this instruction must be used on the SX48/52BD to make sure the
     ; MODE register is written with the correct value. This macro fixes this.
     ; So, instead of using the MODE or MOV M, # instructions to load the M register, use
      _mode instead.
     _mode macro 1
     IFDEF SX48_52
 expand
          mov
               w, \# \setminus 1
                               ; loads the M register correctly for the SX48BD
                               ; and SX52BD
          mov
               m,w
 noexpand
    ELSE
 expand
          mov
               m, \# \setminus 1
                               ; loads the M register correctly for the
                               ;SX18AC, SX20AC
                               ; and SX28AC
 noexpand
    ENDIF
     endm
     ; INCP/DECP macros for incrementing/decrementing pointers to RAM
     ; used to compensate for incompatibilities between SX28 and SX52
     ; Virtual Peripheral Guidelines Tip:
          To support compatibility between source code written for the SX28 and the SX52,
          use macros. This macro compensates for the fact that RAM banks are contiguous
          in the SX52, but separated by 0x20 in the SX18/28.
```

```
TNCP
    macro 1
         inc
              \1
    IFNDEF SX48_52
    setb \1.4
                            ; If SX18 or SX28, keep bit 4 of the pointer = 1
    ENDIF
                            ; to jump from $1f to $30, etc.
{\tt endm}
DECP
    macro 1
    IFDEF SX48 52
         dec \1
    ELSE
              \1.4
                            ; If SX18 or SX28, forces rollover to next bank
         clrb
         dec
              \1
                            ; if it rolls over. (Skips banks with bit 4 = 0)
         setb \1.4
                            ; Eg: $30 --> $20 --> $1f --> $1f
    ENDIF
                        ; AND: $31 --> $21 --> $20 --> $30
endm
    ; Error generating macros
    ; Used to generate an error message if the label is unintentionally moved into the
    ; second half of a page. Use for lookup tables.
    ; Virtual Peripheral Guidelines Tip:
         Surround lookup tables with the tableStart and tableEnd macros. An error will
         be generated on assembly if the table crosses a page boundary.
         Example:
              lookupTable1
                   add
                       pc,w
              tableStart
                       Λ
                   retw
                   retw
                       20
                   retw
                       -20
                   retw
                       -40
              tableEnd
    tableStart
                             ; Generates an error message if code that MUST be in
         macro 0
                             ; the first half of a page is moved into the second
                            ;half.
    if $ & $100
         ERROR 'Must be located in the first half of a page.'
    endif
endm
tableEnd
                            ; Generates an error message if code that MUST be in
         macro 0
                             ; the first half of a page is moved into the second
                            ;half.
    if $ & $100
         ERROR 'Must be located in the first half of a page.'
    endif
endm
```

```
; Data Memory address definitions
; These definitions ensure the proper address is used for banks 0 - 7 for 2K SX devices
; (SX18/20/28) and 4K SX devices (SX48/52).
IFDEF SX48_52
              $0A
global_org
bank0_org
         =
              $00
bank1_org
              $10
bank2_org
              $20
         =
              $30
bank3_org =
bank4_org =
              $40
              $50
bank5_org =
bank6_org =
              $60
              $70
bank7_org =
ELSE
global_org =
              $08
bank0_org
         =
              $10
bank0_org
bank1_org
              $30
bank2_org =
              $50
bank3_org =
              $70
bank4_org =
              $90
bank5_org =
              $B0
bank6_org =
              $D0
              $F0
bank7_org =
ENDIF
; Global Register definitions
; NOTE: Global data memory starts at $0A on SX48/52 and $08 on SX18/20/28.
global_org
         org
     ; Virtual Peripheral Guidelines Tip:
         Use only these defined label types for global registers. If an extra temporary
         register is required, adhere to these label types. For instance, if two
         temporary registers are required for the Interrupt Service Routine,
         use the label isrTemp1 for it.
     flags0
         equ
              ; and function-enabling bits (semaphores)
     ;VP: RS232 Receive
rs232RxFlag equ flags0.0
                             ; indicates the reception of a bit from the UART
flags1
                            ; stores bit-wise operators like flags
         equ
              global_org + 1
                             ; and function-enabling bits (semaphores)
localTemp0
              global_org + 2
                            ; temporary storage register
          equ
                             ; Used by first level of nesting
                             ; Never guaranteed to maintain data
localTemp1
          equ
              global_org + 3
                             ; temporary storage register
                             ; Used by second level of nesting
                             ; or when a routine needs more than one
                             ; temporary global register.
              global_org + 4
localTemp2
                            ; temporary storage register
          eau
```

```
; Used by third level of nesting or by
                         ; main loop routines that need a loop
                         ; counter, etc.
isrTemp0
        equ
            global_org + 5
                         ; Interrupt Service Routine's temp register.
                         ; Don't use this register in the mainline.
; RAM Bank Register definitions
; Bank 0
    org
             bank0_org
bank0
            $
    ; Virtual Peripheral Guidelines Tip:
        - Avoid using bank0 in programs written for SX48/52.
    ; Bank 1
    orq
             bank1_org
    ; Virtual Peripheral Guidelines Tip:
    ;
        Tip 1:
        Indicate which Virtual Peripherals a portion of source code or declaration
        belongs to with a ;VP: VirtualPeripheralName comment.
    ;
        Tip 2:
        All RAM location declaration names should be
        - left justified
        - less than 2 tabs in length
        - written in hungarian notation
        - prefixed by a truncated version of the Virtual Peripheral's name
        Examples:
        ; VP: RS232 Transmit
                          Ś
                                 ;RS232 Transmit bank
        rs232TxBank
    ;
        rs232TxHigh
                     ds
                          1
                                  ; hi byte to transmit
    ;
    ;
        rs232TxLow
                     ds
                          1
                                  ;low byte to transmit
                         1
                                  ;number of bits sent
        rs232TxCount
                     ds
        rs232TxDivide
                     ds
                         1
                                  ;xmit timing (/16) counter
        rs232TxString
                     ds
                          1
                                  ; the address of the string to be sent
        rs232TxByte
                     ds
                          1
                                  ;semi-temporary serial register
    ; VP: ISR Multithreader
isrMultiplex
            ds
                 1
                         ; The isrMultiplex register is used to switch to a
```

```
; new execution thread on each pass of the ISR.
; VP: RS232 Transmit
                                   ;UART Transmit bank
rs232TxBank
rs232TxHigh
                 ds
                                   ; hi byte to transmit
                        1
rs232TxLow
                 ds
                        1
                                   ;low byte to transmit
rs232TxCount
                 ds
                        1
                                   ;number of bits sent
rs232TxDivide
                 ds
                        1
                                   ;xmit timing (/16) counter
;VP: RS232 Receive
rs232RxBank
                         $
                                   ;UART Receive bank
rs232RxCount
                 ds
                        1
                                   ;number of bits received
                 ds
                        1
rs232RxDivide
                                   receive timing counter;
rs232RxByte
                 ds
                        1
                                   ;buffer for incoming byte
                                   ;used by serial routines
rs232Byte
                 ds
      ; Bank 2
      bank2_org
           org
bank2
                  $
      ; Bank 3
                  bank3_org
           org
bank3
                 $
                  bank4_org
           org
bank4
      ; Bank 5
           org
                  bank5_org
bank5
      ; Bank 6
            org
                  bank6_org
bank6
                 $
           org
                  bank7_org
bank7
                 $
```

```
IFDEF SX48_52
   ; Bank 8
           $80
                      ;bank 8 address on SX52
       org
bank8
           $
   ; Virtual Peripheral Guidelines Tip:
       - This extra memory is not available in the SX18/28, so don't use it for Virtual
        Peripherals written for both platforms.
   ; Bank 9
                      ;bank 9 address on SX52
       org
           $90
bank9
           $
   ; Bank A
                      ;bank A address on SX52
       org
           $A0
bankA
           $
   ; Bank B
           $B0
                      ;bank B address on SX52
       ora
bankB
           $
   ; Bank C
       org
           $C0
                      ;bank C address on SX52
bankC
           $
   $D0
                      ;bank D address on SX52
       ora
bankD
           $
   ; Bank E
           $E0
                      ;bank E address on SX52
       org
```

```
bankE
                $
     ; Bank F
     org
                $F0
                                ;bank F address on SX52
                $
bankF
ENDIF
     ; Pin Definitions:
     ; Virtual Peripheral Guidelines Tip:
          - Store all initialization constants for the \ensuremath{\text{I/O}} in the same area, so
            pins can be easily moved around.
          - Pin definitions should follow the same format guidelines as RAM definitions
                - Left justified
                - Hungarian Notation
                - Less that 2 tabs in length
                - Indicate the Virtual Peripheral the pin is used for
          - Only use symbolic names to access a pin/port in the source code.
          - Example:
                ; VP: RS232 Transmit
                     rs232TxPin equ
                                      ra.3
     ; VP: RS232 Receive
rs232RxPin
           equ
                 ra.2
                                ;UART receive input
; VP: RS232 Transmit
rs232TxPin
                                ;UART transmit output
           equ
                ra.3
                %00001000
                                ;SX18/20/28/48/52 port A latch init
RA_latch
           equ
RA_DDIR
                                ;SX18/20/28/48/52 port A DDIR value
           equ
                 %11110111
RA_LVL
                 %00000000
                                ;SX18/20/28/48/52 port A LVL value
           equ
RA_PLP
                 %00001000
                                ;SX18/20/28/48/52 port A PLP value
           equ
RB_latch
           equ
                %0000000
                                ;SX18/20/28/48/52 port B latch init
RB_DDIR
                                ;SX18/20/28/48/52 port B DDIR value
           equ
                %11111111
RB_ST
           equ
                 %11111111
                                ;SX18/20/28/48/52 port B ST value
RB_LVL
                 %00000000
                                ;SX18/20/28/48/52 port B LVL value
           equ
RB_PLP
           equ
                 %00000000
                                ;SX18/20/28/48/52 port B PLP value
RC_latch
           equ
                 %00000000
                                ;SX18/20/28/48/52 port C latch init
                                ;SX18/20/28/48/52 port C DDIR value
RC_DDIR
           equ
                 %11111111
RC_ST
                 %11111111
                                ;SX18/20/28/48/52 port C ST value
           equ
                 %00000000
                                ;SX18/20/28/48/52 port C LVL value
RC_LVL
           equ
                               ;SX18/20/28/48/52 port C PLP value
RC_PLP
                 %00000000
           equ
```

```
IFDEF SX48_52
                               ;SX48BD/52BD Port initialization values
RD_latch equ %00000000
                               ;SX48/52 port D latch init
RD_DDIR
          equ %11111111
                              ;SX48/52 port D DDIR value
RD_ST
          equ %11111111
                              ;SX48/52 port D ST value
         equ %0000000
RD_LVL
                              ;SX48/52 port D LVL value
RD_PLP
          equ %00000000
                              ;SX48/52 port D PLP value
RE_latch
          equ
               %0000000
                              ;SX48/52 port E latch init
                %11111111
RE_DDIR
          equ
                               ;SX48/52 port E DDIR value
                              ;SX48/52 port E ST value
RE_ST
          equ %11111111
RE_LVL
          equ %0000000
                              ;SX48/52 port E LVL value
RE_PLP
          equ %0000000
                              ;SX48/52 port E PLP value
ENDIF
; Program constants
; Virtual Peripheral Guidelines Tip:
          To calculate the interrupt period in cycles:
          - First, choose the desired interrupt frequency
               - Should be a multiple of each Virtual Peripherals sampling frequency.
               - Example: 19200kHz UART sampling rate * 16 = 307.200kHz
          - Next, choose the desired oscillator frequency.
               - 50MHz, for example.
          - Perform the calculation int_period = (osc. frequency / interrupt frequency)
                                    = (50MHz / 307.2kHz)
                                    = 162.7604
          - Round int_period to the nearest integer:
          - Now calculate your actual interrupt rate:
                                    = osc. frequency / int_period
                                    = 50MHz / 163
                                    = 306.748 \text{kHz}
          - This interrupt frequency will be the timebase for all of the Virtual
           Peripherals
     int_period =
               108
                               ; Gives an interrupt period at 50MHz of
                               ; (108 * (1/25000000)s) = 4.32us Which gives
                               ; an interrupt frequency of
                               ; (1/4.34us)Hz = 231481kHz
     ; Virtual Peripheral Guidelines Tip:
          - Include all calculations for Virtual Peripheral constants for any sample
          - Relate all Virtual Peripheral constants to the sample rate of the Virtual
           Peripheral.
          - Example:
               ; VP: 5ms Timer
               TIMER_DIV_CONSTequ 192; This constant = timer sample rate/200Hz = 192
     ; VP: RS232 Transmit AND
```

```
; VP: RS232 Receive
                  57870
UART1_Fs
                               ; Actual calculated ISR frequency / 4.
                               ; How often is the UART sampled? If it is sampled on
                               ; every 4th pass of the ISR, so this number is the
                               ; ISR rate/4 this number must be close to the desired
                               ; UART rate * n. where n must be an even number
                               ; and preferably >= 4
                               ; For instance: For 38400bps, use 38400Hz*4, 38400Hz*6,
                               ;etc.
                  9600
UART1_Baud
                              ; Baud rate is 9600bps
; *** Uart Divide Rates: These numbers indicate the divide rate for the UARTs.
             Example: If the desired UART rate is 19200 and the actual sample
;
                        rate is 230.4kHz, the divide ratio is 230.4kHz/19200Hz = 12
UART1_Divide
                  =
                        UART1_Fs/UART1_Baud ; Divide rate constant used by the program
UART1_St_Delay
                        UART1_Divide + (UART1_Divide/2); Start delay constant used by
                        the program
IFDEF SX48_52
      ;*****************************
      ; SX48BD/52BD Mode addresses
      ; *On SX48BD/52BD, most registers addressed via mode are read and write, with the
      ; exception of CMP and WKPND which do an exchange with W.
      ; Timer (read) addresses
TCPL_R
       equ $00
                                     ;Read Timer Capture register low byte
          equ
TCPH_R
                  $01
                                     ;Read Timer Capture register high byte
TR2CML_R
          equ
                  $02
                                     ;Read Timer R2 low byte
TR2CMH_R equ
                $03
                                    ;Read Timer R2 high byte
TR1CML_R
          equ $04
                                    ;Read Timer R1 low byte
TR1CMH_R
                  $05
                                    ;Read Timer R1 high byte
          equ
TCNTB_R
                  $06
                                    ;Read Timer control register B
            equ
TCNTA_R
                  $07
                                    ;Read Timer control register A
          equ
; Exchange addresses
CMP
          equ
                  $08
                                     ;Exchange Comparator enable/status register with W
WKPND
           equ
                  $09
                                     ; Exchange MIWU/RB Interrupts pending with W
; Port setup (read) addresses
WKED_R
       equ
                  $0A
                                     ;Read MIWU/RB Interrupt edge setup,
                                     ;0 = falling, 1 = rising
       equ
                  $0B
                                     ;Read MIWU/RB Interrupt edge setup,
WKEN R
                                     ;0 = enabled, 1 = disabled
ST_R
                  $0C
                                     ; Read Port Schmitt Trigger setup,
            equ
                                     ;0 = enabled, 1 = disabled
LVL_R
            equ
                  $0D
                                    ;Read Port Level setup, 0 = CMOS, 1 = TTL
PLP_R
                  $0E
                                     ;Read Port Weak Pullup setup,
            equ
                                     ;0 = enabled, 1 = disabled
DDIR_R
            equ
                  $0F
                                     ;Read Port Direction
; Timer (write) addresses
TR2CML_W
                  $12
                                     ;Write Timer R2 low byte
          equ
                                     ;Write Timer R2 high byte
TR2CMH_W
                  $13
            equ
TR1CML_W
            equ $14
                                    ;Write Timer R1 low byte
TR1CMH_W
                  $15
                                    ;Write Timer R1 high byte
            equ
TCNTB_W
                                    ;Write Timer control register B
            equ
                  $16
TCNTA_W
                  $17
                                    ;Write Timer control register A
            equ
```

```
; Port setup (write) addresses
                               ;Write MIWU/RB Interrupt edge setup,
WKED_W
          equ
               $1A
                               ;0 = falling, 1 = rising
                               ;Write MIWU/RB Interrupt edge setup,
WKEN_W
               $1B
          equ
                               ;0 = enabled, 1 = disabled
ST_W
                               ;Write Port Schmitt Trigger setup,
          equ
               $1C
                               ;0 = enabled, 1 = disabled
LVL_W
          equ
               $1D
                               ;Write Port Level setup, 0 = CMOS, 1 = TTL
PLP_W
          equ
               $1E
                               ; Write Port Weak Pullup setup, 0 = enabled,
                               ;1 = disabled
DDIR_W
               $1F
                               ;Write Port Direction
          eau
ELSE
     ; SX18AC/20AC/28AC Mode addresses
     ; *On SX18/20/28, all registers addressed via mode are write only, with the exception
     ; of CMP and WKPND which do an exchange with W.
     ; Exchange addresses
               $08
                               ; Exchange Comparator enable/status register with W
CMP
          equ
WKPND
               $09
                               ; Exchange MIWU/RB Interrupts pending with W
          equ
; Port setup (read) addresses
                               ;Write MIWU/RB Interrupt edge setup,
WKED_W
          equ
               $0A
                               ;0 = falling, 1 = rising
                               ;Write MIWU/RB Interrupt edge setup,
WKEN_W
               $0B
          equ
                               ;0 = enabled, 1 = disabled
ST_W
                               ;Write Port Schmitt Trigger setup,
          equ
               $0C
                               ;0 = enabled, 1 = disabled
                               ;Write Port Schmitt Trigger setup,
LVL_W
          equ
               $0D
                               ;0 = enabled, 1 = disabled
                               ;Write Port Schmitt Trigger setup,
PLP_W
               SOE
          equ
                               ;0 = enabled, 1 = disabled
               $0F
                               ;Write Port Direction
DDIR W
          equ
ENDIF
; Program memory ORG defines
; Virtual Peripheral Guidelines Tip:
          - Place a table at the top of the source with the starting addresses of all of
           the components of the program.
     INTERRUPT_ORG
               equ
                               ; Interrupt must always start at location zero
RESET_ENTRY_ORG
                     $1FB
                              ; The program will jump here on reset.
               equ
SUBROUTINES_ORG
                     $200
                              ; The subroutines are in this location
               equ
STRINGS_ORG
                    $300
                               ; The strings are in location $300
               equ
PAGE3_ORG
               equ
                    $400
                               ; Page 3 is empty
MAIN_PROGRAM_ORG
               equ
                    $600
                               ; The main program is in the last page of program
                               ; memory.
```

```
INTERRUPT_ORG
                             ; First location in program memory.
; Interrupt Service Routine
;_________
; Note: The interrupt code must always originate at address $0.
; Interrupt Frequency = (Cycle Frequency / -(retiw value)) For example:
; With a retiw value of -163 and an oscillator frequency of 50 \text{MHz}, this
; code runs every 3.26us.
                        73 The interrupt service routine...
ISR
;-----
; VP: VP Multitasker
    ; Virtual Peripheral Guidelines Tip:
     ; - Multi-thread the Interrupt Service Routine
          - Produces a FAR smaller worst-case cycle time count, and enables a larger
          number of VP's to run simultaneously. Also produces "empty" slots that future
          VP's can be copied and pasted into easily.
         - Determine how often your tasks need to run. (9600bps UART can run well at a
           sampling rate of only 38400Hz, so don't run it faster than this.)
         - Strategically place each "module" into the threads of the ISR. If a module
           must be run more often, just call it's module at double the rate or quadruple
           the rate, etc....
         - Split complicated Virtual Peripherals into several modules, keeping the
          high-speed portions of the Virtual Peripherals as small and quick as possible,
          and run the more complicated, slower processing part of the Virtual Peripheral
           at a lower rate.
     ;______
     ; Virtual Peripheral Multitasker: up to 24 individual threads, each running at
                          the interrupt rate/24. Change the
         Input variable(s): isr_multiplex: variable used to choose threads
         Output variable(s): None, executes the next thread
         Variable(s) affected: isr_multiplex
         Flag(s) affected: None
```

```
Program Cycles: 9 cycles (turbo mode)
     ;-----
     bank
                isrMultiplex
                                ; 1
               isrMultiplex
                                ; 1
                                           ; toggle interrupt rates
                w,isrMultiplex
                                ; 1
     mov
; The code between the tableBegin and tableEnd statements MUST be
; completely within the first half of a page. The routines
; it is jumping to must be in the same page as this table.
tableStart
                                ; Start all tables with this macro.
     jmp
                pc+w
                                ; 3
                isrThread1
                                ;3,9 cycles. isrThread1 runs the UART
     jmp
                isrThread2
     jmp
                isrThread3
     jmp
     jmp
                isrThread4
                                ; Call this thread 4 times/cycle for an execution
     jmp
                isrThread1
                                ;rate of ISR_rate/4
     jmp
                isrThread5
     jmp
               isrThread6
                                ;
     jmp
                isrThread7
               isrThread1
     qmj
     jmp
               isrThread8
               isrThread9
     jmp
               isrThread10
     jmp
               isrThread1
     jmp
                isrThread11
     jmp
                isrThread12
     qmţ
     qmį
                isrThread13
tableEnd
                                ; End all tables with this macro.
;-----
; VP: VP Multitasker
; ISR TASKS
                ; Serviced at ISR rate / 4
isrThread1
; Virtual Peripheral Guidelines Tip:
          The sample rate of this section of code is the isr rate / 4, because it is jumped
          to in every 4th entry in the VP Multitaskers table. To increase the
           sample rate, put more calls to this thread in the Multitasker's jump table.
     ; VP: RS232 Transmit
;-----
; Virtual Peripheral: Universal Asynchronous Receiver Transmitter (UART)
; These routines send and receive RS232 serial data, and are currently
; configured (though modifications can be made) for the popular
; "No parity-checking, 8 data bit, 1 stop bit" (N,8,1) data format.
; TRANSMITTING: The transmit routine requires the data to be inverted
; and loaded (tx_high+tx_low) register pair (with the inverted 8 data bits
; stored in tx_high and tx_low bit 7 set high to act as a start bit). Then
; the number of bits ready for transmission (10=1 start + 8 data + 1 stop)
; must be loaded into the tx_count register. As soon as this latter is done,
; the transmit routine immediately begins sending the data.
; This routine has a varying execution rate and therefore should always be
; placed after any timing-critical virtual peripherals such as timers,
; adcs, pwms, etc.
; Note: The transmit and receive routines are independent and either may be
      removed, if not needed, to reduce execution time and memory usage,
```

```
as long as the initial "BANK serial" (common) instruction is kept.
;
       Input variable(s) : tx_low (only high bit used), tx_high, tx_count
      Variable(s) affected : tx_divide
      Program cycles: 17 worst case
      Variable Length? Yes.
:______
rs232Transmit
            _bank rs232TxBank
                                   ;2 switch to serial register bank
            decsz rs232TxDivide ;1 only execute the transmit routine
            jmp
                 :rs232TxOut
                                   ; 1
                 w,#UART1_Divide
                                   ;1 load UART baud rate (50MHz)
            mov
            mov
                 rs232TxDivide,w
                                   ; 1
            test rs232TxCount
                                   ;1 are we sending?
                                   ; 1
            snz
                 :rs232TxOut
                                   ; 1
            jmp
:txbit
                                    ;1 yes, ready stop bit
            clc
            rr
                 rs232TxHigh
                                    ;1 and shift to next bit
            rr
                 rs232TxLow
                                    ; 1
            dec
                 rs232TxCount
                                   ;1 decrement bit counter
            snb
                 rs232TxLow.6
                                   ;1 output next bit
            clrb rs232TxPin
                                   ; 1
            sb
                 rs232TxLow.6
                                   ; 1
            setb rs232TxPin
                                   ;1,17
:rs232TxOut
; VP: RS232 Receive
;______
; Virtual Peripheral: Universal Asynchronous Receiver Transmitter (UART)
; These routines send and receive RS232 serial data, and are currently
; configured (though modifications can be made) for the popular
; "No parity-checking, 8 data bit, 1 stop bit" (N,8,1) data format.
; RECEIVING: The rx_flag is set high whenever a valid byte of data has been
; received and it is the calling routine's responsibility to reset this flag
; once the incoming data has been collected.
       Output variable(s) : rx_flag, rx_byte
       Variable(s) affected : tx_divide, rx_divide, rx_count
      Flag(s) affected : rx_flag
      Program cycles: 23 worst case
      Variable Length? Yes.
rs232Receive
            _bank rs232RxBank
                                   ; 2
             sb
                 rs232RxPin
                                   ;1 get current rx bit
             clc
                                    ; 1
             snb rs232RxPin
                                    ;1
             stc
                                    ;1
            test rs232RxCount
                                   ;1 currently receiving byte?
            SZ
                 :rxbit
                                   ;1 if so, jump ahead
            ami
                 w,#9
                                   ;1 in case start, ready 9 bits
            mov
            sc
                                   ;1 skip ahead if not start bit
                 rs232RxCount,w ;1 it is, so renew bit count
            mov
                 w,#UART1_St_Delay ;1 ready 1.5 bit periods (50MHz)
            mov
                 rs232RxDivide,w
            mov
                                    ; 1
            decsz rs232RxDivide
:rxbit
                                    ;1 middle of next bit?
                                   ;1
                 :rs232RxOut
            jmp
                 w,#UART1_Divide
                                   ;1 yes, ready 1 bit period (50MHz)
            mov
```

```
rs232RxDivide,w
           mov
                                 ; 1
           dec rs232RxCount
                                 ;1 last bit?
                                  ;1 if not
           SZ
               rs232RxByte
                                ;1 then save bit
           rr
                                  ;1 if so,
           snz
           setb rs232RxFlag
                                 ;1,23 then set flag
:rs232RxOut
                                  ; else, exit
           jmp isrOut ;7 cycles until mainline program resumes execution
```

```
;-----
isrThread2
                ; Serviced at ISR rate / 16
;-----
   jmp isrOut
                ;7 cycles until mainline program resumes execution
;-----
isrThread3
               ; Serviced at ISR rate / 16
;-----
   jmp isrOut
               ;7 cycles until mainline program resumes execution
;-----
isrThread4
               ; Serviced at ISR rate / 16
   jmp isrOut
                ;7 cycles until mainline program resumes execution
:______
isrThread5
                ; Serviced at ISR rate / 16
     isr0ut
                ;7 cycles until mainline program resumes execution
   jmp
:-----
isrThread6
                ; Serviced at ISR rate / 16
;-----
   jmp
     isrOut
                ;7 cycles until mainline program resumes execution
;-----
                ; Serviced at ISR rate / 16
isrThread7
;-----
   jmp isrOut
                ;7 cycles until mainline program resumes execution
;-----
isrThread8
                ; Serviced at ISR rate / 16
:______
   jmp isrOut
                ;7 cycles until mainline program resumes execution
isrThread9
                ; Serviced at ISR rate / 16
jmp
  isrOut
                ;7 cycles until mainline program resumes execution
;-----
isrThread10
                ; Serviced at ISR rate / 16
;-----
   jmp isrOut
               ;7 cycles until mainline program resumes execution
```

```
;-----
              ; Serviced at ISR rate / 16
   jmp isrOut
                  ;7 cycles until mainline program resumes execution
                  ; Serviced at ISR rate / 16
;______
   ami
       isrOut
                  ;7 cycles until mainline program resumes execution
isrThread13
                  ; Serviced at ISR rate / 16
                  ; This thread must reload the isrMultiplex register
           ; since it is the last one to run in a rotation.
   bank isrMultiplex
   mov isrMultiplex, #255
                     ; Reload isrMultiplex so isrThread1 will be run
                      ; on next interrupt.
   jmp isrOut
isrOut
;-----
   mov w,#-int_period ;1 ; return and add -int_period to the RTCC
   retiw
                  ; 3
                     ; using the retiw instruction.
RESET_ENTRY_ORG
; Virtual Peripheral Guidelines Tip:
       The main program operation should be easy to find, so place it at the end of the
       program code. This means that if the first page is used for anything other than
       main program source code, a reset_entry must be placed in the first page, along
       with a 'page' instruction and a 'jump' instruction to the beginning of the
       main program.
   resetEntry
                      ; Program starts here on power-up
   page _resetEntry
   jmp _resetEntry
; Virtual Peripheral Guidelines Tip:
       ORG statements should use predefined labels rather than literal values.
```

SUBROUTINES_ORG

ora

```
; Subroutines
; VP: RS232 Transmit
; Function: send_byte
; Send byte via serial port
; INPUTS:
     W
               The byte to be sent via RS-232
; OUTPUTS:
; outputs the byte via RS-232
;-----
         mov
sendByte
               localTemp0,w
          _bank rs232TxBank
         test rs232TxCount
                                     ; wait for not busy
:wait
          SZ
          jmp
               :wait
          not
                                      ;ready bits (inverse logic)
          mov
                rs232TxHigh,w
                                     ;store data byte
          setb
               rs232TxLow.7
                                     ;set up start bit
                                     ;1 start + 8 data + 1 stop bit
          mov
                w,#10
          mov
                rs232TxCount,w
          retp
                                     ; leave and fix page bits
; VP: RS232 Transmit
; Subroutine - Send string pointed to by address in W register
; INPUTS:
                The address of a null-terminated string in program
                memory
; OUTPUTS:
   outputs the string via. RS-232
sendString
          _bank rs232TxBank
          mov
               localTemp1,w
                                     ;store string address
:loop
          mov
               w, #STRINGS_ORG>>8
                                     ; with indirect addressing
          mov
               m,w
                w,localTemp1
                                     ;read next string character
          mov
                                      ;using the mode register
          iread
          test
                                      ; are we at the last char?
                                     ;if not=0, skip ahead
          snz
          qmj
               :out
                                     ;yes, leave & fix page bits
          call
                sendByte
                                     ;not 0, so send character
          _bank rs232TxBank
          inc
               localTemp1
                                     ;point to next character
          jmp
                :loop
                                      ;loop until done
:out
          mov
                w, #$1F
                                      ;reset the mode register
          mov
                m,w
          retp
                 ______
; VP: RS232 Receive
; Subroutine - Get byte via serial port.
; INPUTS:
    -NONE
; OUTPUTS:
```

```
-received byte in rs232Byte and w register
;-----
getByte
       jnb
          rs232RxFlag,$
                        ; wait till byte is received
       clrb rs232RxFlag
                        reset the receive flag
       _bank rs232RxBank
                        ;switch to rs232 bank
       mov rs232Byte,rs232RxByte ;store byte (copy using W)
       retp
STRINGS ORG
             ; This label defines where strings are kept in program space.
; String Data
; VP: RS232 Transmit
_hello
       dw 13,10,'Yup, The UART works!!!',0
_hitSpace
       dw 13,10,'Hit Space...',0
   ; Virtual Peripheral Guidelines Tip:
       - Routines that use location-dependant data, such as in example below, should
        use a LABEL rather than a literal value as their input. Example:
       instead of
              m,#3 ; move upper nybble of address of strings into m
          mov
        use
          mov m,#STRINGS_ORG>>8; move upper nybble of address of strings into m
   PAGE3 ORG
; Virtual Peripheral Guidelines Tip:
       To ensure that several Virtual Peripherals, when pasted together, do not cross
       a page boundary without the integrator's knowledge, put an ORG statement and
       one instruction at every page boundary. This will generate an error if a pasted
       subroutine moves another subroutine to a page boundary.
   ; This instruction will cause an assembler error if the source code
   jmp
          ; before the org statement inadvertantly crosses a page boundary.
MAIN_PROGRAM_ORG
```

```
; RESET VECTOR
       ; Program execution begins here on power-up or after a reset
_resetEntry
       ; Initialize all port configuration
             _mode
                      ST_W
                                          ;point MODE to write ST register
                                          ;Setup RB Schmitt Trigger, 0 = enabled, 1 = disabled
             mov
                      w, #RB_ST
                      !rb,w
             mov
                                          ;Setup RC Schmitt Trigger, 0 = enabled, 1 = disabled
                      w,#RC_ST
             mov
                      !rc,w
             mov
IFDEF SX48_52
              mov
                      w, #RD_ST
                                          ;Setup RD Schmitt Trigger, 0 = enabled, 1 = disabled
             mov
                      !rd,w
             mov
                      w, #RE_ST
                                          ;Setup RE Schmitt Trigger, 0 = enabled, 1 = disabled
             mov
                      !re,w
ENDIF
              _mode
                      LVL_W
                                          ;point MODE to write LVL register
             mov
                      w, #RA_LVL
                                          ;Setup RA CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
                      !ra,w
                      w, #RB_LVL
                                          ; Setup RB CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
             mov
                      !rb.w
                      w, #RC_LVL
                                          ;Setup RC CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
             mov
                      !rc,w
IFDEF SX48_52
                                          ;Setup RD CMOS or TTL levels, 0 = TTL, 1 = CMOS
                      w, #RD_LVL
             mov
                      !rd,w
             mov
                      w,#RE_LVL
                                          ;Setup RE CMOS or TTL levels, 0 = TTL, 1 = CMOS
             mov
                      !re,w
             mov
ENDIF
                      PLP_W
                                          ;point MODE to write PLP register
              mode
             mov
                      w, #RA_PLP
                                          ;Setup RA Weak Pull-up, 0 = enabled, 1 = disabled
             mov
                      !ra,w
                                          ;Setup RB Weak Pull-up, 0 = enabled, 1 = disabled
                      w,#RB_PLP
             mov
             mov
                      !rb,w
                                          ;Setup RC Weak Pull-up, 0 = enabled, 1 = disabled
             mov
                      w, #RC_PLP
              mov
                      !rc,w
IFDEF SX48_52
                      w, #RD_PLP
                                          ;Setup RD Weak Pull-up, 0 = enabled, 1 = disabled
             mov
                      !rd.w
             mov
                      w, #RE_PLP
                                          ;Setup RE Weak Pull-up, 0 = enabled, 1 = disabled
             mov
             mov
                      !re,w
ENDIF
                                          ;point MODE to write DDIR register
              _mode
                      DDIR_W
                      w, #RA_DDIR
                                          ;Setup RA Direction register, 0 = output, 1 = input
             mov
                      !ra,w
             mov
                      w,#RB_DDIR
                                          ;Setup RB Direction register, 0 = output, 1 = input
             mov
             mov
                      !rb,w
              mov
                      w, #RC_DDIR
                                          ;Setup RC Direction register, 0 = output, 1 = input
             mov
                      !rc,w
IFDEF SX48_52
                      w, #RD_DDIR
                                          ;Setup RD Direction register, 0 = output, 1 = input
             mov
                      !rd,w
             mov
```

```
w, #RE_DDIR
                                         ;Setup RE Direction register, 0 = output, 1 = input
             mov
             mov
                      !re,w
ENDIF
                      w, #RA_latch
                                         ; Initialize RA data latch
             mov
             mov
                      ra,w
                      w,#RB_latch
                                         ;Initialize RB data latch
             mov
             mov
                      rb,w
             mov
                      w,#RC_latch
                                         ; Initialize RC data latch
             mov
                      rc,w
IFDEF SX48_52
                                        ;Initialize RD data latch
             mov
                      w,#RD_latch
             mov
                      rd.w
                      w,#RE_latch
                                         ;Initialize RE data latch
             mov
             mov
                      re,w
ENDIF
       ; Clear all Data RAM locations
zeroRam
IFDEF SX48_52
                                         ;SX48/52 RAM clear routine
             mov
                     w,#$0a
                                         ;reset all ram starting at $0A
             mov
                     fsr,w
:zeroRam
             clr
                     ind
                                         ; clear using indirect addressing
                                         ;repeat until done
             incsz
                     fsr
              jmp
                     :zeroRam
                     bank0
             bank
                                         ;clear bank 0 registers
             clr
                     $10
             clr
                     $11
             clr
                     $12
                     $13
             clr
             clr
                     $14
             clr
                     $15
             clr
                     $16
             clr
                     $17
             clr
                     $18
             clr
                     $19
             clr
                     $1a
             clr
                     $1b
             clr
                     $1c
             clr
                     $1d
             clr
                      $1e
             clr
                     $1f
ELSE
                                         ;SX18/20/28 RAM clear routine
             clr
                     fsr
                                         ;reset all ram banks
:zeroRam
             sb
                     fsr.4
                                         ; are we on low half of bank?
             setb
                     fsr.3
                                         ;If so, don't touch regs 0-7
             clr
                     ind
                                         ; clear using indirect addressing
              incsz
                                         ;repeat until done
                     fsr
              jmp
                      :zeroRam
ENDIF
       ; Initialize program/VP registers
             bank
                     rs232TxBank
                                        ; Select the bank
```

```
w,#UART1_Divide
                                    ;load initial UART baud rate
            mov
                   rs232TxDivide,w
            mov
      ; Setup and enable RTCC interrupt, WREG register, RTCC/WDT prescaler
      ; Virtual Peripheral Guidelines Tip:
            The suggested default values for the option register are:
      ;
            - Bit 7 set to 0: location $01 addresses the W register (WREG
            - Bit 5 set to 1: RTCC increments on internal transitions
            - Bit 3 set to 1: Prescaler assigned to WatchDog Timer
            If a routine must change the value of the option register (for example, to
            access the RTCC register directly), then it should restore the default value
            for the option register before exiting.
      RTCC_ON
                  %10000000
                                     ; Enables RTCC at address $01 (RTW hi)
                                     ; *WREG at address $01 (RTW lo) by default
                                     ;Disables RTCC edge interrupt (RTE_IE hi)
RTCC_ID
                  %01000000
                                     ;*RTCC edge interrupt (RTE_IE lo)
                                     ; enabled by default
RTCC_INC_EXT =
                                     ; Sets RTCC increment on RTCC pin transition (RTS hi)
                  %00100000
                                     ;*RTCC increment on internal instruction (RTS lo)
                                     is default;
RTCC FE
                  %00010000
                                     ;Sets RTCC to increment on falling edge (RTE_ES hi)
                                     ;*RTCC to increment on rising edge (RTE_ES lo) is
                                     ;default
                  %00000000
RTCC_PS_ON
                                    ; Assigns prescaler to RTCC (PSA lo)
            =
RTCC_PS_OFF =
                  %00001000
                                    ;Assigns prescaler to WDT (PSA lo)
PS_000
                  %00000000
                                    ;RTCC = 1:2, WDT = 1:1
PS_001
                  %00000001
                                    ;RTCC = 1:4, WDT = 1:2
            =
                                    ; RTCC = 1:8, WDT = 1:4
PS_010
           =
                 %00000010
PS_011
                                    ;RTCC = 1:16, WDT = 1:8
            =
                 %00000011
PS_100
                                    ;RTCC = 1:32, WDT = 1:16
            =
                 %00000100
                                    ;RTCC = 1:64, WDT = 1:32
PS_101
                  %00000101
                                    ;RTCC = 1:128, WDT = 1:64
PS_110
                  %00000110
                                    ;RTCC = 1:256, WDT = 1:128
PS_111
            =
                  %00000111
                  RTCC_PS_OFF | PS_111 ; the default option setup for this program.
OPTIONSETUP
            equ
                  w,#OPTIONSETUP ; setup option register for RTCC interrupts enabled
            mov
            mov
                  !option,w
                                    ; and no prescaler.
                  @mainLoop
            jmp
; MAIN PROGRAM CODE
            mov
                  w,_hitSpace
                                           ; Send prompt to terminal at UART rate
            call
                  @sendString
mainLoop
            call
                  @getByte
                  rs232Byte,#' ',mainLoop ; just keep looping until user
                                           ; hits the space bar
            mov
                  w,#_hello
                                           ; When space bar hit, send out string.
```

	call jmp	esendString mainLoop		
;*****************************				
END ; End of				
; * * * * * * * * * * *	*****	*********************		
; * * * * * * * * * * *	*****	*******************		
; * * * * * * * * * * *	*****	*********************		
; * * * * * * * * * * *	*****	********************		
; * * * * * * * * * * *	*****	********************		
; * * * * * * * * * * *	*****	********************		
; * * * * * * * * * * *	*****	********************		